NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

GRAPHIC INTERFACE FOR ATTRIBUTE-BASED DATA LANGUAGE QUERIES FROM A PERSONAL COMPUTER TO A MULTI-LINGUAL, MULTI-MODEL, MULTI-BACKEND DATABASE SYSTEM OVER AN ETHERNET NETWORK

by

William Goebel Anthony Sympson III

December 1989

Thesis Advisor: C. Thomas Wu

Approved for public release; distribution is unlimited

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification UNCLASSIFIED	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report
2b Declassification/Downgrading Schedule	Approved for public release; distribution is unlimited.
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (If Applicable): 52	7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (city, state, and ZIP code): Monterey, CA 93943-5000	7b Address (city, state, and ZIP code): Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization; 8b Office Symbol (If Applicable)	9 Procurement Instrument Identification Number
8c Address (city, state, and ZIP code)	10 Source of Funding Numbers

Program Element Number	Project No	Task No	Work Unit Accession No

11 Title (Include Security Classification) GRAPHIC INTERFACE FOR ATTRIBUTE-BASED DATA LANGUAGE QUERIES FROM A PERSONAL COMPUTER TO THE MULTI-LINGUAL, MULTI-MODEL, MULTI-BACKEND DATABASE SYSTEM OVER AN ETHERNET NETWORK

12 Personal Author(s) Sympson III, William G. A.

13a Type of Report	13b Time Covered		14 Date of Report (year, month, day)	15 Page Count
Master's Thesis	From	To	1989 December	128

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (continue on reverse if necessary and identify by block number)
Field	Group	Subgroup	object-oriented language, graphical interface, database language

19 Abstract (continue on reverse if necessary and identify by block number

This research is aimed at taking one step closer to the goal of the "paperless ship". This thesis examined the easibility of providing a visual interface to allow queries from a front end Personal Computer (PC) using the Attribute-Based Data Language (ABDL) to a Multi-lingual, Multi-model, Multi-backend mini-computer Providing n improved Human-Machine Interface for the system will greatly increase its usability. A prototype was mplemented in the Graphics LAnguage for Database (GLAD) on a Zenith 248 as the front end connected to a ISI nini-computer running the Multi-Lingual, Multi-Model, Multi-Backend Database System (MBDS), a backend of ne future. The Zenith 248 was chosen as the front end because of the large quantity of these computers nroughout the Navy. GLAD was used because it is a graphics object-oriented environment for databases that ives the user access to both data manipulation and program development through visual interaction. This creates user friendly windowing environment both for development and for operational applications. Looking towards ne future, MBDS is the perfect backend as it is the latest in Database management systems. This thesis provided n extension to GLAD to demonstrate the ability to send Attribute-Based Data Language to Multi-Backend Database System .

20 Distribution/Availability of Abstract	21 Abstract Security Classification
[X] unclassified/unlimited [ ] same as report [ ] DTIC users	Unclassified

22a Name of Responsible Individual	22b Telephone (Include Area code)	22c Office Symbol
Professor C. Thomas Wu	(408) 646-3391	Code 52 Wq

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted

All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

Graphic Interface for Attribute-Based Data Language Queries from a Personal Computer to the Multi-Lingual, Multi-Model, Multi-Backend Database System Over an Ethernet Network

by

William Goebel Anthony Sympson III
Lieutenant, United States Navy
B.S.S.E., United States Naval Academy, 1984

Submitted in partial fulfillment of the requirements for
the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1989

ABSTRACT

This research is aimed at taking one step closer to the goal of the "paperless ship". This thesis examined the feasibility of providing a visual interface to allow queries from a front end **P**ersonal **C**omputer (**PC**) using the **A**ttribute-**B**ased **D**ata **L**anguage (**ABDL**) to a Multi-lingual, Multi-model, Multi-backend mini-computer Providing an improved Human-Machine Interface for the system will greatly increase its usability. A prototype was implemented in the **G**raphics **LA**nguage for **D**atabase (**GLAD**) on a Zenith 248 as the front end connected to a ISI mini-computer running the Multi-Lingual, Multi-Model, **M**ulti-**B**ackend **D**atabase **S**ystem (**MBDS**), a backend of the future. The Zenith 248 was chosen as the front end because of the large quantity of these computers throughout the Navy. GLAD was used because it is a graphics object-oriented environment for databases that gives the user access to both data manipulation and program development through visual interaction. This creates a user friendly windowing environment both for development and for operational applications. Looking towards the future, MBDS is the perfect backend as it is the latest in Database management systems. This thesis provided an extension to GLAD to demonstrate the ability to send Attribute-Based Data Language to Multi-Backend Database System .

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to express my sincere thanks to the people who helped me prepare and implement this thesis. I would like to thank my advisor Dr. C. Thomas Wu for his help in the conception and preparation of this thesis.

I would also like to thank my family, particularly my beautiful wife Susan for her patience, support and love, without which none of this would have been possible. And my children, Julianne, Christopher, and William who make each day brighter than the day before.

I. INTRODUCTION

A. BACKGROUND

As the 90's begin, we are at the dawn of the "**paperless ship**" concept. This concept was first made popular by a former director of Surface Warfare, VADM J. Metcalf, USN (Retired). Studies have determined that an Oliver Hazard Perry guided missile frigate (FFG-7 class) with a crew of 185 men, and nominal 3500 ton displacement carries in excess of 20 tons of paper required for the ship's mission and crew [Ref. 1:pp. 157-159]. The goal of the "paperless ship" is to reduce and ultimately eliminate tons of paper. This reduction would allow the installation of more offensive or defensive weaponry to improve all ship's ability to fulfill its required mission. It would also greatly reduce the manpower and man-hours required to maintain this paperwork. This thesis will present a tool using existing hardware and software to make the "paperless ship" a reality tomorrow.

Tactical computers have been in the Navy for decades. Non-tactical computers have only been in service on board ships for the last 15 years. The largest program at present is the Ship's Non-tactical ADP program (**SNAP II**). While SNAP II was a giant leap forward toward reaching the goal of the "paperless ship", it fell far short of its mark. At times it seems to add paperwork. The SNAP II used hardware of the 60's with software of the 70's and didn't reach the majority of the ships until the mid 80's. We need something better for the 90's.

On the average Destroyer, the SNAP II system consists of one Harris mini computer with a storage capacity of 80 Mbytes hooked up to 6 to 8 dumb terminals. The software is still being added to, but some of the programs it handles are supply requisition and

inventory, Personnel Qualification System, and several simple tickler files. It also has a limited word processing ability. It takes lots of man-hours to properly train individuals on the basic functions of the system with each section requiring its own specific lessons. With several terminals in use at the same time, which is a normal situation on most ships, the systems performance degrades drastically.

The second major step forward occurred in 1985 when Zenith Data Corporation was awarded a DOD contract to provide PC's, Zenith 248 AT compatible computers for the Defense Department. Before that time, there was no standardization which led to a lot of incompatibility of equipment. Also, for a command to purchase a computer it had to write a letter of justification which took six months to get routed and approved. Then it took an additional six months for Zenith and the supply system to deliver the units. After the standardization, and when supply was able to keep up with the high demands, ships and squadrons were able to receive several units.

B. THE NEED FOR RESEARCH

The situation as we enter the 90's is that a destroyer class ship has a SNAP II system installed with 8 terminals and 5 to 7 Zenith 248 computers which are stand alone units. This Thesis will propose that we can greatly improve the ship's ability to :

- reduce paperwork
- reduce man hours
- increase utilization and performance of the SNAP II system
- increase utilization and performance of the Zenith 248's
- by use of a graphical interface to simplify procedures for all users

Using the Zenith 248's as the front end would greatly reduce the load on the SNAP II system since a majority of the work could be done on the 248's own processor, and only

access the SNAP's for specific information. The 248, using commercially developed software, can handle a majority of the programs that are now done on SNAPs. With more terminals available, time will not be wasted waiting for an open one. Using GLAD provides an interface where the user is able to directly manipulate the required data. Given the diversity of users, direct manipulation of logical objects is appealing to novices, easy to remember for intermittent users and rapid for frequent users. [Ref. 2].

The focus of this research was to test the feasibility of providing such a graphical interface from a front-end, in this case, a Zenith 248, and a backend database such as SNAP II or its replacement.

C. THESIS ORGANIZATION

Chapter II provides a discussion of Object oriented programing and the Actor software development environment.

Chapter III provides a description of the Multi-Lingual, Multi-Model, Multi-Backend Database System (MBDS) and its Attribute Based Data Language. It discusses the connection between the Backend and the Front end through a socket interface design by another thesis student.

Chapter IV explains the implementation details of the query interface. First it looks at the Graphics Language for Database (GLAD) background, and the reason it was chosen as the interface for use in the system is discussed. Then it describes the modifications and improvements to GLAD. And the specific implementation of the QueryWindow, the sending of ABDL Queries and the data it receives back from MBDS , as well as some of the associated implementation difficulties are presented. And lastly, a sample session of the research results running in GLAD is given. The various windows that can be manipulated are illustrated and described.

Chapter V concludes with a summary discussion of the research and possible enhancements as continuation of research in this area.

II. OBJECT-ORIENTED PROGRAMMING AND THE ACTOR ENVIRONMENT

A. OBJECT-ORIENTED PROGRAMMING (OOP)

1. Introduction

Object-oriented programming is a new and powerful programming environment. Rather than working with traditional procedures, subroutines and separate data structures, the programmer creates objects which control both the data structure and the operations on that data. One of the goals of object-oriented programming languages is to reduce the coding required to be written and maintained. This is accomplished by allowing the programmer to build classes of reusable objects which encapsulate behavior and ensure data abstraction.

Many languages claim to be object-oriented languages, however, in order for a language to be considered object-oriented it must meet three criteria [Ref. 3:p. 1]:

- encapsulation of data and instructions into units of functionality called objects
- inheritance of functionality through a class hierarchy
- dynamic run-time binding of messages sent to objects

Languages such as Ada, Modula-2, and C++ which are advertised as object-oriented languages fail to meet all three criteria. The Actor environment provides for *message-passing paradigm, inheritance, and polymorphism,* the major characteristics of object-oriented languages. Actor is an object-oriented programming language (OOL) for MS-DOS microcomputers from The Whitewater Group Inc. The Actor environment will be discussed in detail later in this chapter.

The basis of object-oriented programming is the creation and management of objects. This is obtained by using the fundamental concepts in any OOL which are *object, class,* and *message.*

2. Fundamental concepts of Object-Oriented Language

a. Objects

An *object* is a programming entity that resembles both tangible and intangible real-world objects. An object has attributes and responds to instructions. At first it is difficult to grasp the concept of an object. But we deal with objects constantly (people, cars, pictures), OOP represents these physical objects as computer objects. For example, an object as a functional entity is a car. A car object has attributes such as model, engine, year and color. A car also responds to certain actions or instructions, such as go (step on the gas), stop (step on the brake), turn (turn the wheel). When you want the car to go, you are not interested in the drive train or the engine. All cars are driven the same way, regardless of model or year. This is called *data abstraction*, the ability to manipulate an object's data without knowledge of the data's internal format. The big difference between procedural languages and object-oriented languages is that in procedural functions work on data passed to them as parameters, while in object-oriented you send a message to the objects to perform operations on themselves. In Actor, many entities normally considered data structures are actually objects. For example, integers, characters, strings and files are considered objects.

b. Classes

A *class* defines the structure and behavior of an object. A class describes what is common in a category of objects, such as: the class of ships or class of students. Any class describes a set of objects, where these objects are called *instances* of a class. The instructions that an object can respond to are managed by the class, while the data

associated with a particular object is managed by the object itself. Using the previous car example, Dodge is a class of car, and all Dodge cars belong to the class, but you might have a red Dodge Daytona and a blue Dodge Caravan. The structure of a class is defined by *instance variables* which contain the data private to that class. The behavior of an object of a class is defined by its *methods*. Methods allow other objects to access this data.

c. Messages

Objects perform operations in response to *messages*. An object responds to a *message* based on a behavior defined by the object's *methods*. Again back in the car, when we press on the gas pedal, we send a go message to the car object. The car object's transmission system has defined methods to respond to the message, increase fuel, increase engine rpm.

There are two types of *methods*, what we have been using are called *object methods*. These are associated to the instances of the class, and only allow messages to that instance of the class. We need to be able to send messages not to the instances but to the class itself. The best example of this is the *new* method to create a new object of a class, it can't be an *object method* because the object does not exist, so it is handle by a *class method*.

How does Actor execute *methods* and *messages?* A *method* gets executed by sending a *message* to an object. The format for a message in Actor consists of a *selector,* a *receiver,* and *a list of arguments*. When a message is sent to an object, it looks to see if a method of the same name as the *selector* exists, if so, it executes the method. If it fails to find a method within the instance of that class, an error message is generated. The syntax of a method definition in Actor is:

```
/* Comments */
Def <methodName>(self [argument list [| <local variables>]])
{       statement 1;
        statement 2;
                .
                .
                .
        statement n;
}!!
```

Actor methods can take up to eight arguments, and up to eight local variables. The local variables are assigned and exist only during the life of the method. The above method can be executed by sending it a message:

```
methodName( Receiver, arguments);
```

The *selector* would be "methodName", the *receiver* is "Receiver" , and the *list of arguments* "arguments".

3. Inheritance, Polymorphism and Encapsulation

a. Inheritance

Inheritance is a mechanism for sharing behaviors between classes. In Actor, each class inherits behavior from classes above it, its *ancestors,* and passes down behavior to classes below it, its *descendants*. This behavior means that a descendant class has access to all its ancestor's instance variables and methods, in addition to its own. Actor uses *single inheritance* where each class is permitted only one direct ancestor. [Ref. 4:p. 39]

This inheritance works throughout the Actor *class tree,* which is the hierarchical ordering of all its classes. The most generic classes are at the top and the

more specialized classes at the bottom. All new classes must be descendants of classes already defined in Actor. Since all classes are descendants of **Object** class, all classes inherit methods and instance variables defined for the **Object** class.

When a message is sent to an object, the methods defined for the class which this object belongs to are first searched. If there are no corresponding methods, then the methods defined for the object's immediate ancestor class, parent class, are searched. If again the search fails, then the methods of the object's grandparent are searched. This is repeated all the way to the **Object** class.

In our previous example , the car class defines how cars in general behave. The class Dodge inherits the general car behavior from car class, and adds behavior that is specific to Dodge cars. But not all Dodges are the same so we define a Caravan class which inherits from first Dodge class, then car class. Assume all Dodge's use the same transmission, so that method would be defined in the Dodge class. When we press on the gas pedal of a Caravan, a go message is sent. The Caravan class does not define a go method, so the search continues to Caravan's immediate ancestor class, Dodge class. The go method of Dodge is then executed.

b. Polymorphism

Polymorphism literally is defined as "the ability to take several forms". In OOL, it is used to describe a situation where the same message causes different responses depending on who the receiver of the message is. Actor allows different classes to have the same method names. The methods could also have totally different implementations. The result is that we can send the same message to different objects to produce different responses. Polymorphism allows us to write generic reusable code more easily, since we can specify general instructions and delegate the implementation details to the objects

that are involved. This decreases the dependencies in the code and maintenance is therefore easier.

In the Dodge class, we can define a basic stereo method which says all Dodges have AM/FM radios. In the Caravan we can define a separate stereo method that states that all caravans have AM/FM cassette radios. The program determines at run-time who the correct *receiver* of the message should be so that the correct stereo method is executed.

c. Encapsulation

One of the main goals of object-oriented programming is *encapsulation.* It is accomplished by allowing access to data only through its own methods. No other parts of the program can operate directly on another object's data. Therefore, this ensures that the proper instructions are operated on. This allows many objects to respond to the same messages but will execute their own methods. A program can send a generic message and leave the implementation up to the receiving object. This decreases interdependencies, and improves interchangeability and reusability.

One last look at our car example, we can think of a car's brake system as being encapsulated. Although brakes may differ in implementation, disc or anti-lock, they are all used by the driver by using the pedal: step on the brake pedal to stop the car, then let go of the brake pedal and keep moving . It does not matter what type of brakes a car has, that detail is insulated from the rest of the car and the driver. That makes it easy for a driver to be able to use any car.

4. Object-Oriented Programming Summary

Procedural programming and object-oriented programming are quite different, and understanding OOP was the first obstacle to overcome to implement this thesis. Object-oriented programming closely links data and procedures as objects. The main

reason for using object-oriented programming is the ability to reuse code and develop more maintainable systems in a shorter amount of time.

B. ACTOR ENVIRONMENT

1. Microsoft Windows

Since Actor is a Microsoft Windows (MS-Windows) application, to use it we must know how to operate Windows in order to use it. Windows is a visual extension of MS-DOS that layers itself upon DOS to provide the user with a friendly, graphical interface. See Figure 2-1 for a graphic depiction of this concept. In Windows, the user can execute multiple programs simultaneously in an integrated environment, which gives the user a consistent interface whatever the application. Windows does not require the

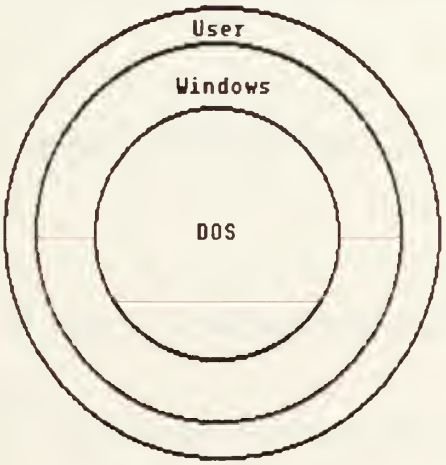

Figure 2.1 Windows layered over DOS

users to memorize command line Commands and their syntax. This reduces the learning curve for all window applications dramatically.[Ref . 5: p. 4]

a. MS-Windows requirements

MS-Windows runs on IBM personal computers or compatibles. Any system on which you install Windows must meet the following minimum requirements:

A personal computer with two floppy disk drives or a fixed hard drive.

512K of memory (640K or greater is recommended)

DOS version 2.0 or greater

A monochrome or color monitor with graphics card

Windows can be operated without a mouse, however, we will describe Windows operations using a mouse, because Actor requires one.

b. Sample Window

Figure 2.2 shows a typical Notepad editor window, in MS-Windows

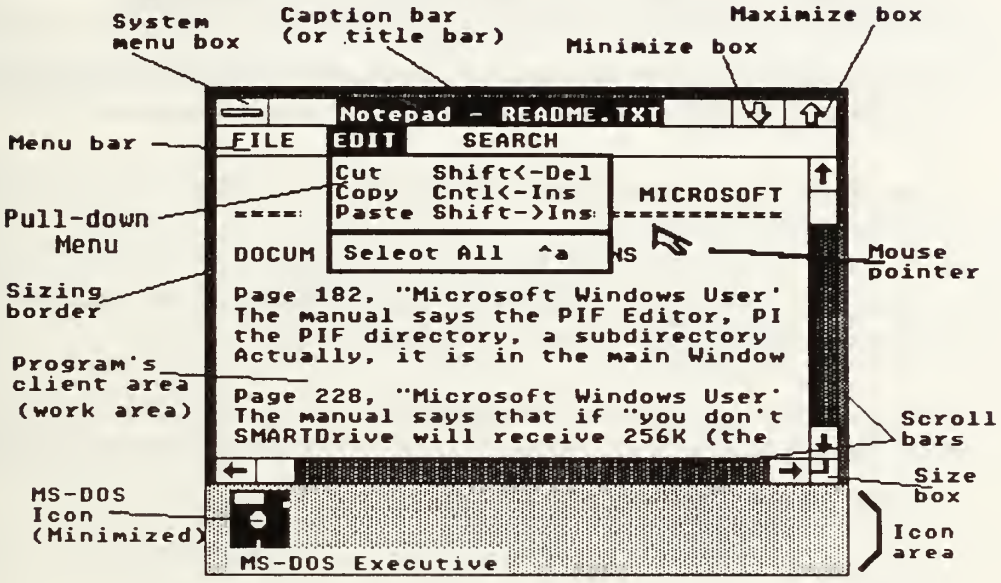

Figure 2.2 Sample Window

environment. Each element of Figure 2 .2 will be briefly described. [Ref. 6:p. 7]

- The *Mouse pointer* will show-up as an arrow pointer.
- The *Program's client area* (Work area) is the working area of the window. We can edit text or graphics by using the keyboard or the mouse.
- The Icons representing applications that have been minimized by clicking on the *minimize box*, located in the right hand corner, appear in the *Icon area* at the bottom of the screen.
- The *Caption bar* (title bar) displays the name of the application in the window. In Figure 2-2, it is the Notepad. If the window is active, the area to

the left and right of the title will be filled in with a color (if capable) or grayed to show that this is where you are working.

- The *System Menu box* can be used to display the System menus in the applications. This menu is common to all Windows programs.
- The *size box* can adjust the size of the window by clicking on it with the mouse and dragging it to either increase or decrease the window.
- The *Menu bar* is similar to the System Menu bar, but this controls the features unique to this window.
- A *Pull-down Menu* is a submenu of an item displayed in the Menu bar. It helps organize features and reduces clutter.
- The *Maximize box* increases the window to fill the entire screen, while the *minimize box* reduces the window to a Icon representation at the bottom of the screen.
- *Scroll bars* appear when there is more than one screen of information to be displayed.
- A vertical line | , I- Beam, indicates that text input is required in that specific part of the window.

c. Dialog Box

For short, simple communications between a Window application and a user, a special window called a *dialog box* is used. Figure 2-3, shows a sample dialog box from PCPaint. Dialogs are used to get specific information required by the application or to warn the user of some error. They may contain several types of control devices, for example;

- push buttons
- radio buttons
- check buttons
- edit fields

- list boxes

MS-Windows provides a visual interface in the form of windows that contain graphic representations of user input and system output. The Menus are the principal means of

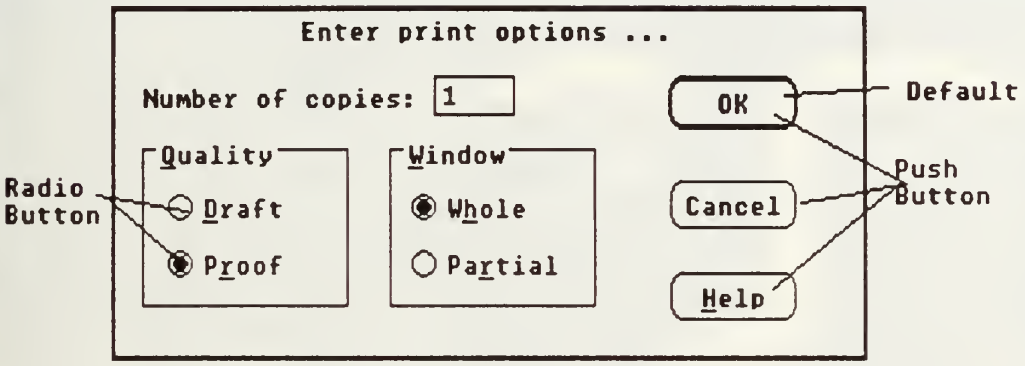

Figure 2.3 Sample Dialog Box

presenting the user options within an application. MS-Windows provides the programmer over 600 "Kit routines" to develop application programs to run under its user interface for its software development.

2. Actor

Actor is a MS-Window application; but unlike other Window applications, Actor provides its source code in the form of predefined classes. This allows programmers to access the Actor functions as well as all the MS-Window's functions. Actor provides both the user and the programmer a very friendly environment, since the MS-Windows operating environment is used . It is much more suitable for programming MS-Windows than Microsoft C. With Actor's predefined classes, a programmer can write just a couple

of lines in Actor that would take literally several pages of C code. This type of coding allows rapid prototyping and testing of applications. Actor provides several tools for software development, which will be discussed later in this chapter.

The first thing you will see upon entering the Actor environment is Figure 2.4, the About Actor box. By clicking on this box with the mouse, we enter the world of

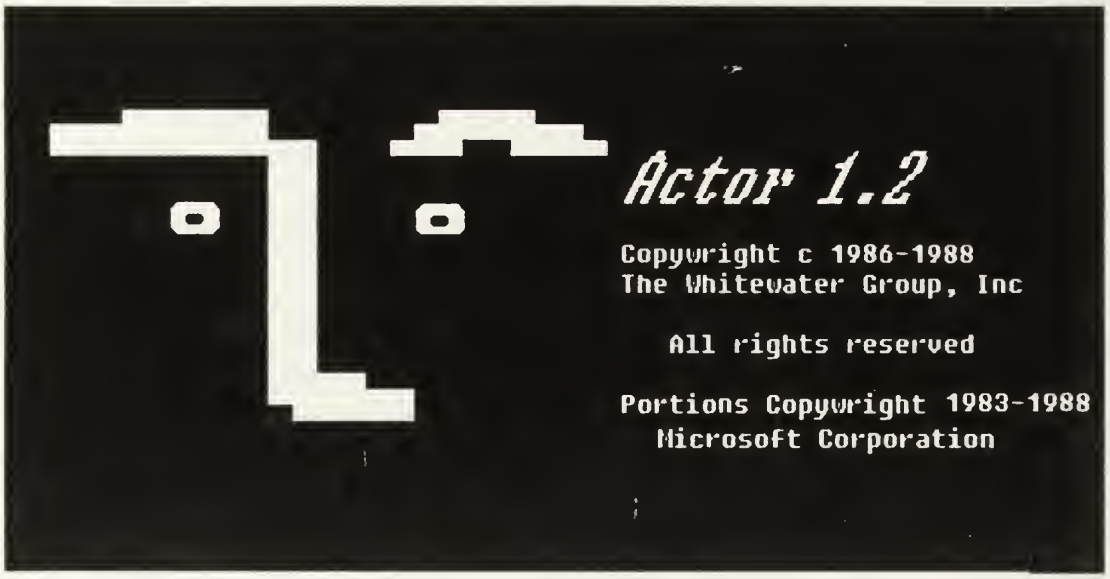

Figure 2.4 The Actor About Box

object-oriented programming of Actor and can take advantage of the rich interactive programming of MS-Windows.

a. Actor requirements

Actor version 1.2 requires:

- All the requirements of MS-Windows plus
- 640K of memory
- a hard disk
- graphics display and adapter
- mouse or other pointing device

This thesis research was successfully tested on a Department of Defense's standard contract Zenith 248 microcomputer in its standard configuration.

b. Starting Actor

When we first begin Actor, two windows come up, The *Display* and the *Workspace*. The *Display* window is used by the Actor environment to print system messages, especially error messages. The *Workspace* window is the heart of the environment. It is here that the programmer brings up other windows, such as the *Browser ,or, Inspection,* and perform system level commands; *Doit!, Cleanup!*. These windows and the system commands will be discussed later.

c. Workspace Window

The Workspace is the main window of Actor. Figure 2.5 shows the Workspace window. There are ten menu choices. The selections with exclamation point means their are no pull-down submenus. The File, Edit, Utility, Templates menu items all have submenus. Actor uses static and dynamic memory, the **Show Room!** menu displays current memory usage. The **Cleanup!** menu item initiates a garbage collection function. The **Doit!** menu item will execute a single command or a series of highlighted commands. The **Inspect!** and **Browse!** open the Inspector and Browser windows

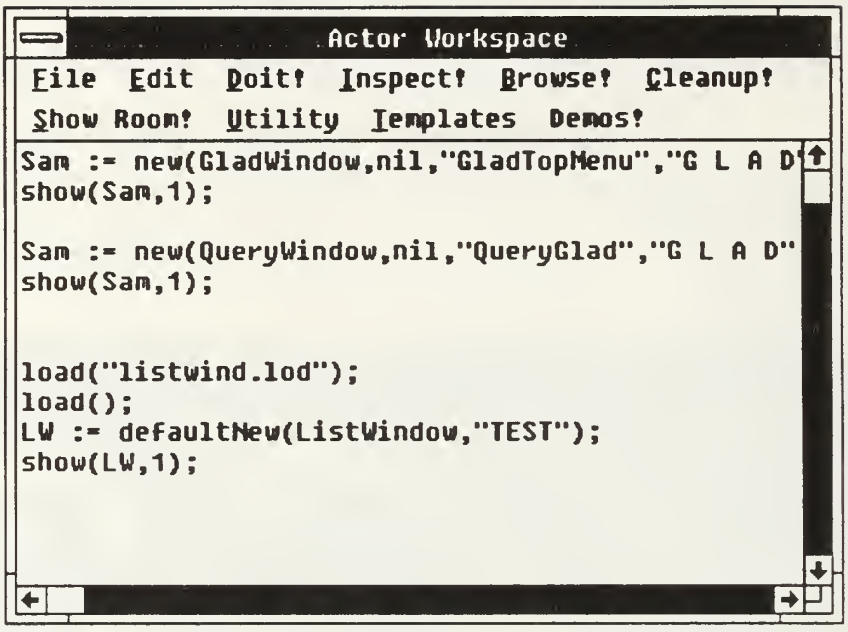

Figure 2.5 Actor Workspace Window

respectfully. Both windows will be discussed in the following section. The **File** choice includes commands concerning files, saving and editing work, and running other MS-Windows programs. The **Edit** provides the ability to cut, copy, paste and clear text. The Edit menu choice reappears in several of the other windows. The **Utility!** menu item gives you methods to search through Actor code to find out which classes define or call a particular method. The **Templates** allows the programmer to pick a control structure like *do, if/then,* and *case*. It then places that template into the Workspace where the programmer can fill in the rest of the code.

d. Browser Window

To open a *Browser* window, click once on the Browser! in the above mentioned Workspace menu bar. The *Browser* window, Figure 2.6, is used to create, modify or destroy classes. It is through this window that we gain access to the powers of Actor. The Browser is where the programmer does most of his work. This window allows us to

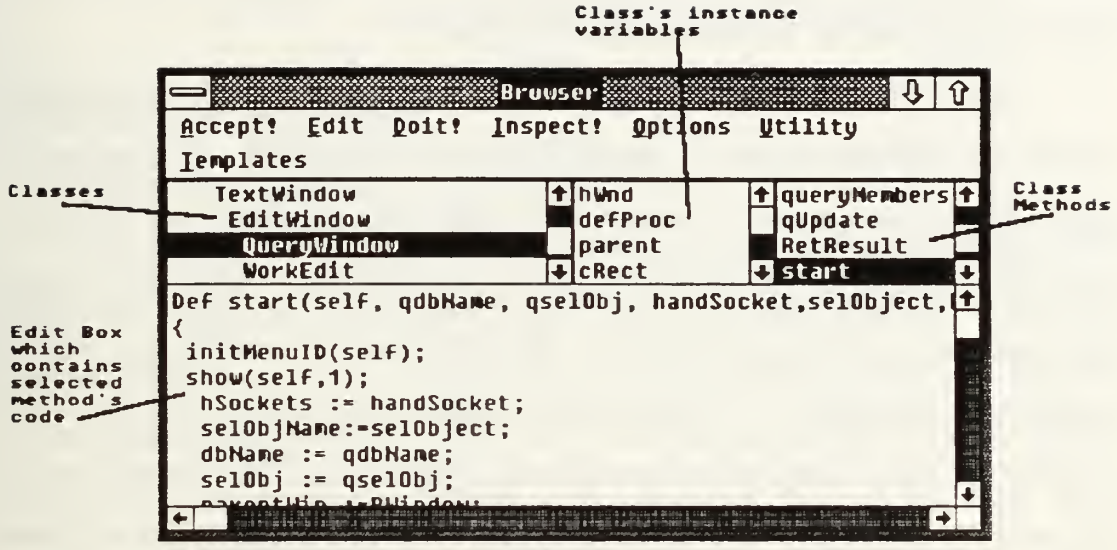

Figure 2.6 The Browser window

examine, edit and add to Actor source code, and in the process, Actor is changed to reflect any changes in the code. The Browser is a specialized file editor designed specifically for manipulating the class source files. Figure 2.6 shows a sample Browser window, the class that has been selected is the QueryWindow class which is the main class for this thesis. This class has several methods that allow a user to formulate ABDL queries and send them to the Backend data server, but this will be discussed in more detail in the next chapters. The method being edited is the start method and its code is then placed in the edit box where it can be added to or modified. [Ref 7:p. 4-15]

e. The Inspector and Debug Windows

The *Inspector* is used to examine the value of objects. To use it, select or highlight any object in the *Workspace* and then click on the menu item **Inspect!** . The Inspector window then comes up. In the upper left corner, we have a list box showing

names of the named instance variables. In the right corner, the keys to the indexed data is displayed. We can select any instance variable and its value will be displayed in the edit box at the bottom of the window. [Ref. 4:p. 1.3.5]

The *Debug window* is an important tool for the programmer. It combines the features from the Inspector and the Browser to allow for diagnosing and correcting programming errors "on the fly". It allows for modification of the program while it is running and will resume the operations with the new corrections. The Browser window catches syntax errors during compile time. The Debug window identifies run time errors and allows the programmer to correct the error while still in the running program. This is a great time saver, being able to accept (compile) and resume. The programmer can place breaks in his program to do some debugging. When the program reaches a break command it initiates the Debug window. [Ref. 4:p. 1.3.7]

C. SUMMARY

The goal of this chapter was to briefly introduce some of the key terminology used in object-oriented programming, and to familiarize ourselves with the Actor environment. Objects, classes, messages and methods are the building blocks in OOP, and are used by Actor to create programs. Actor runs under MS-Windows and provides the programmer its source code to develop applications under a user-friendly software development environment.

III. BACKEND DATA SYSTEM AND SOCKET INTERFACE

A. BACKEND DATA SYSTEM

1. Database Management System DBMS

Over the past twenty five years, many different data models have been developed, starting with traditional data models(such as the relational, the hierarchical and the network models) to the newer, semantic data models(such as functional data model and the entity-relationship model). This led to the implementation of several different DBMS models and languages. Different database models could not use other models data. In fact, updating equipment or a modification of the DBMS software would cause previous transactions to be unusable. In the past couple of years, the ability to network has increased the need for a new all encompassing DBMS. Users can access data or databases around the globe, but because of the many different and unfamiliar languages and models, they can't use the data unless they have people and equipment set up for the specific model .*Multi-lingual, Multi-Model, Multi-Backend Database System* provides a solution: this system allows the user to use/query any DBMS regardless of the specific model, and without concern for the data manipulation language. For example, it will allow a query of a network database via SQL transactions.

a. Multi-Lingual Database System

(MLDS) is a single database system that can execute transactions written respectfully in different data languages and support the structure of various data models. It is able to support all the different data models and languages, with a single underlying database system. This system is referred to as the kernel data model (KDM) and the kernel data language (KDL). Figure 3.1 shows the system structure of the MLDS. The

user interacts with the system through the language interface layer (LIL), using a chosen *user data model* (UDM) to issue transactions written in a corresponding

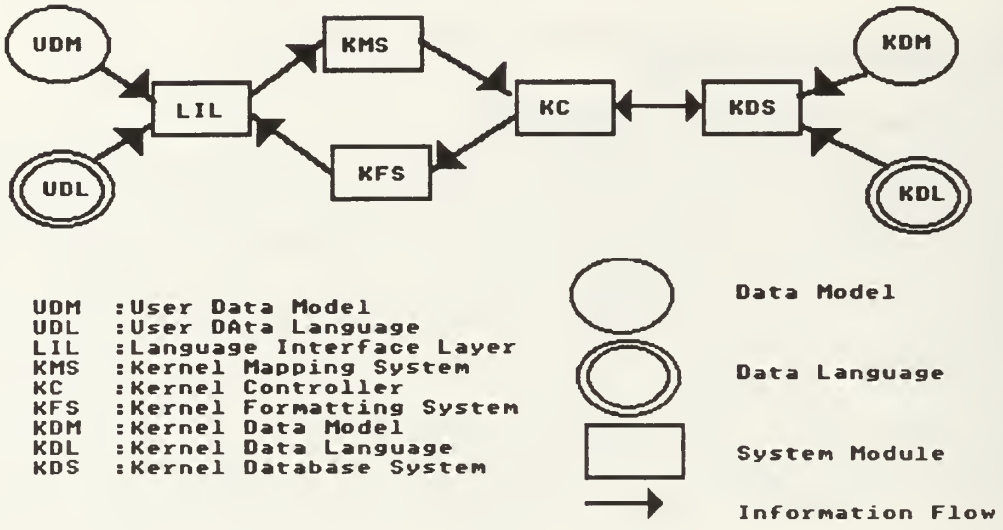

Figure 3.1 The Multi-Linqual Database System [Ref. 8:p.12]

model-based *user data language* (UDL). The LIL sends the user transaction to the *kernel mapping system* (KMS) . The KMS executes one of two possible tasks. The KMS transforms a UDM-based database definition to a database definition of the *kernel data model* (KDM), when the user specifies that a new database is to be created. When the user specifies a UDL transaction is to be executed, the KMS translates the UDL transaction to a transaction in the *Kernel data language* (KDL) equivalents.

In the first task, KMS forwards transaction in the KDM data definition to the *kernel controller* (KC). KC, in turn, sends the KDM database definition to the *kernel database system* (KDS). When KDS is finished with processing the KDM database

definition, it informs the KC. KC then notifies the user, via the LIL, that the database definition has been processed and that loading of the database records may begin.

In the second task, KMS sends the KDL transactions to the KC. When the KC receives the KDL transactions, it forwards them to the KDS for execution. Upon completion, the KDS sends the results in the KDM form back to the KC. The KC routes the results to the *kernel formatting system* (KFS). KFS reformats the results from the KDM form to the UDM form. The KFS then displays the results in the correct UDM form via LIL. The LIL, KMS, KFS, and KC define the language interface for a single user-defined data model. In a MLDS, a separate language interface is required for each model defined. This is shown in figure 3.2.

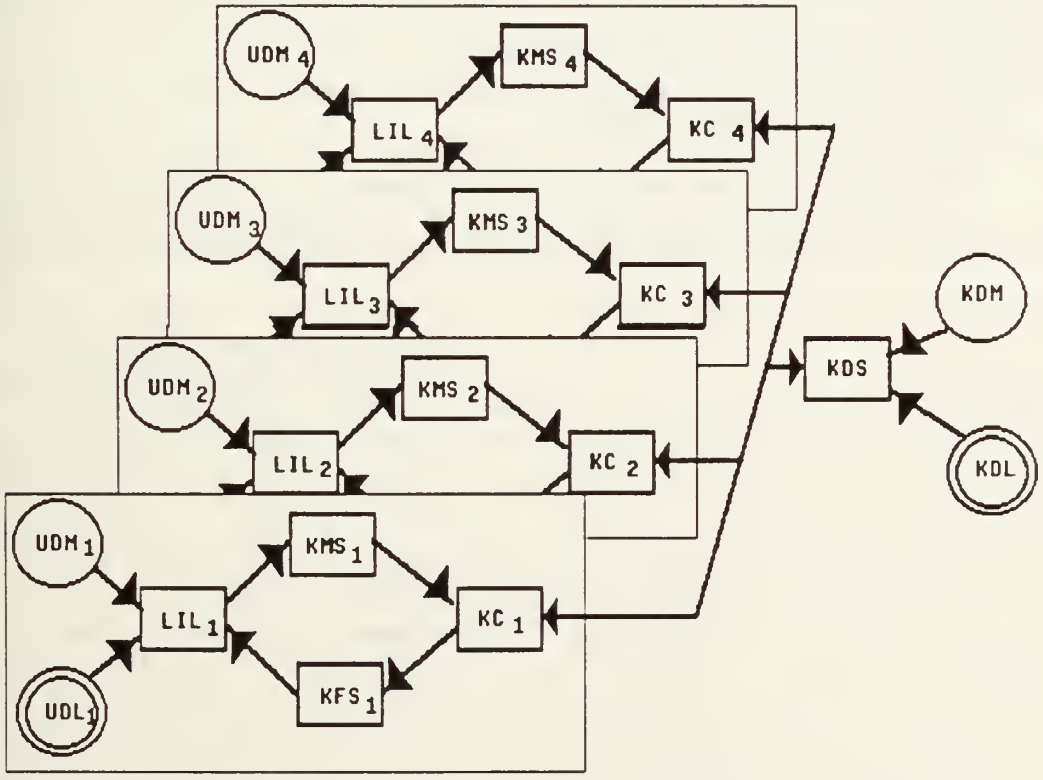

Figure 3.2 Multiple Language Interface

In the current system, there are four unique language interfaces defined.

- relational/SQL model
- hierarchical/ DL/I model
- network/CODASYL-DML model
- functional/Daplex model

These have been developed and implemented by previous thesis students under the guidance of Prof. David Hsiao and his assistant Mr. Thomas Chu. In contrast, the KDS structure is a single, common component shared by all models. The KDS allows the various user-defined language interfaces to access and manipulate the physical database. The attribute-based data model and attribute-based data language (ABDL) have been implemented as the KDM and KDL, respectively, for the MLDS. The ABDL will be discussed later in this chapter. [Ref. 9:p. 11-13]

b. Multi-Model Database System

The *multi-model database system* is an improvement on the MLDS. It allows a user to query any database using a data manipulation language of his choice, regardless of the underlying data model. An example might be, a user can utilize a SQL query transaction to access a hierarchical database. This allows existing databases and query languages to be used when upgrading to a new database system. No longer do we have to retrain people or rewrite the database every time we expand. This reduces error caused during translation of existing database and existing queries into new systems.

The *mixed-processing* strategy is implemented to carry out the cross-access of the databases as shown in Figure 3.3. Two components are involved, the schema transformer and a second language interface. When a user selects a database that is not in the local LI, language interface, all other LI's are searched for the desired database. Upon finding the database, the original database schema is copied and transformed into

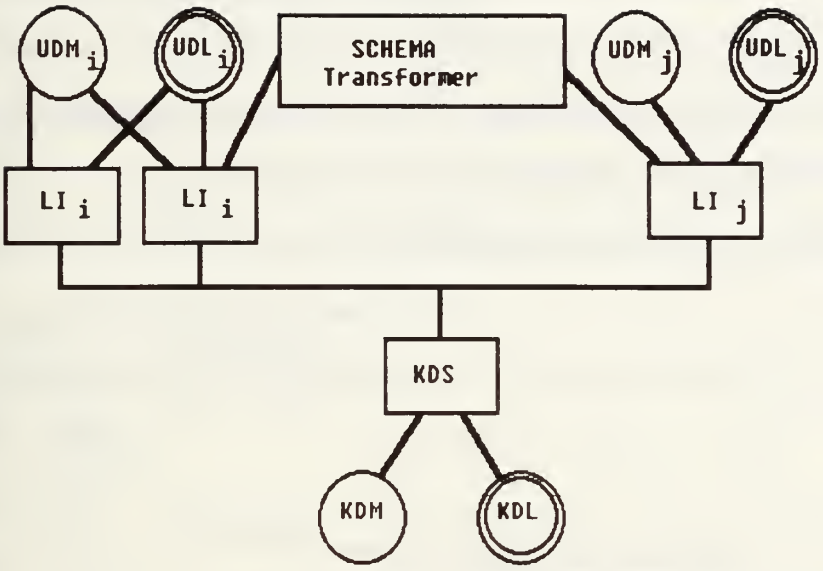

Figure 3.3 The Mixed-Processing Strategy

an equivalent schema in the local LI. When a user executes a transaction in the local data manipulation language, the new language interface processes the request. The attribute based request's output from this language interface is in the form of the original database model which thereby eliminates the need for an extra language translation step. [Ref. 8: p. 34]

c. Multi-Backend Database System

The *Multi-backend database system* overcomes the performance problems and upgrade issues related to the traditional approach of database system design. This is accomplished through the utilization of multiple backends connected in parallel fashion. These backends have identical hardware, replicated software, and their own disk systems. A backend controller is responsible for supervising the execution of the database transactions and for interfacing with the hosts and user, see Figure 3.4 . The backends perform the database operations with the database stored on the disk system of the backends.

As Shown in Figure 3.4, the user access is accomplished through a host computer to the controller. When a transaction (a single request or a sequence of requests) is received, the controller broadcasts the transaction to all the backends. Since the database is distributed across the backends, all the backends processors execute the same request in parallel. Each backend maintains its own request queue. As soon as a

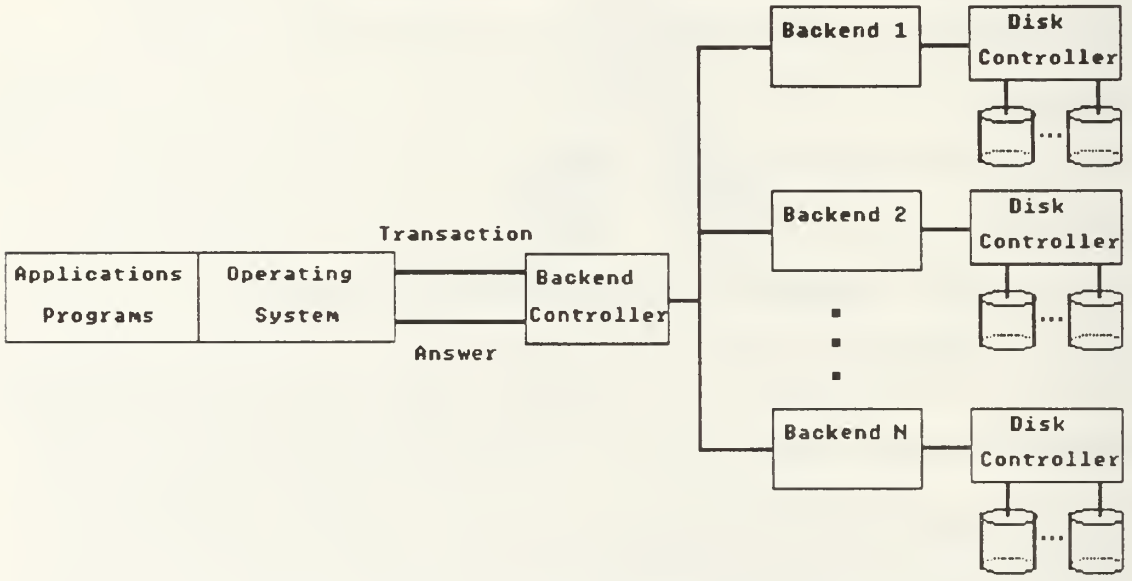

Figure 3.4 The Multi-Backend Database System

backend finishes a request, it sends the result back to the controller and continues to process the next request independent of the other backends. [Ref. 10:p. 32-33]

Performance gains are obtained by increasing the number of backends. For example, if the size of the database and the transactions remain constant, then MBDS would decrease the response time for a user transaction when the number of backends increased. Even more important, if the database or the transactions increased in size, MBDS could maintain an invariant response time with proportional increase in backends.

2. The Data Model and Data Language

The goal of the data-model transformation and data-language translations is to maintain *semantic preservation* of the database and *operational equivalence* of the transactions. As stated above, the attribute-based data model and ABDL have been implemented as the KDM and KDL respectively in the multi-lingual database system.

a. The Attribute-Based Data Model

In the *attribute-based data model,* the data has the following constructs:

- Database — consists of a collection of files
- File — contains a group of records which are characterized by a unique set of keywords
- Record — is made up of a collection of attribute-value pairs
- Attribute-value pairs — is a member of the Cartesian product of the attribute name and the value domain of the attribute.

For example, <model,"Caravan"> is an attribute-value pair having "Caravan" as the value for the model attribute. A record contains at most one attribute-value pair for each attribute defined in the database. Certain attribute-value pairs of a record are called the *directory keywords* of the record, because either the attribute-value pairs or their attribute-value ranges are kept in a *directory* for identifying the records. Those attribute-value pairs which are not kept in the directory are called *non-directory keywords*. An example of a record is shown below.

(<FILE,USCars>,<MAKE,Dodge>,<MODEL,Caravan>,{mini-van})

The angle brackets, <,>, enclose an attribute-value pair, i.e., keyword. The curly brackets , {,}, include the record body. The first attribute-value pair of all records contains the attribute FILE and its value is the file name, in this case, USCars.

There are two major reasons for choosing the attribute-based data model. First, the attribute-based data model is *data independent*, by this we mean, *implementation and application independent*. All of the constructs mentioned above are not dependent on a specific implementation or application. Second, the model allows the user to take advantage of certain constructs for system optimization.

b. The Attribute-Based Data Language

The *attribute-based data language* (ABDL) supports five primary database operations, **INSERT, DELETE, UPDATE, RETRIEVE,** and **RETRIEVE-COMMON.** A *request* is the primary operation in ABDL. Each request contains a *qualification* which is used to specify the part of the database that is to be operated on. Two or more requests may be grouped together to form a *transaction*. The five operations will be shown below. It is through these transactions that we are able to query and modify the database. These are very important to this thesis, their importance will be shown in chapter four.

The **INSERT** request is used to add a new record into an existing database. The qualification of an INSERT request is a list of attribute-value pairs and record body being inserted. For example, the following INSERT request

INSERT(<FILE,USCars>,<MAKE,Ford>,<MODEL,Tempo>)

will insert a record into the USCars file for the Make Ford with a Model Tempo. This record does not contain a record body.

The **UPDATE** request is used to modify records of a database. There are two parts of the qualification of the UPDATE request. The two parts are the query and the modifier. The query specifies which records of the database are to be modified. And the modifier specifies how the records being updated are to be modified. For example, the following UPDATE request

UPDATE (FILE=USCars)(MODELYEAR=90)

will modify all of the records of the USCars file by changing all the model years to 90. In the above request, the (FILE=USCars) is the query and (MODELYEAR=90) is the modifier. Another example for when you only want a specific part of your database updated is shown below.

UPDATE((FILE=USCars)and(MAKE=AMC))(MAKE=Chrysler)

This example will change all the records in USCars of make, AMC, to replace AMC with Chrysler. The query is ((FILE=USCars)and(MAKE=AMC)), note the placement of parenthesis, and the modifier is (MAKE=Chrysler).

The **DELETE** is used to remove one or more records from the the database. The qualification of a DELETE is a query. For example, the following request

DELETE((FILE=USCars)and(MAKE=Edsel))

will delete all records whose make is Edsel in the USCars file.

The **RETRIEVE** request is used to access records of the database. the qualification of a retrieve consist of a query, a target-list, and a by-clause. Again the query specifies which records are to be retrieved. The *target-list* consists of a list of attributes whose attribute values are to be output to the user. The optional *by-clause* may be used to group records. The following RETRIEVE request

RETRIEVE((FILE=USCars) and(MAKE=Chrysler))(MODEL)

will output to the user the model names of all the records in the USCars file with a make of Chrysler. In the above example the query was ((FILE=USCars) and(MAKE=Chrysler)), the target-list was (MODEL),and the by-clause was not used. Here is a RETRIEVE request using the by-clause.

RETRIEVE(FILE=USCars)(MAKE,MODEL,MODELYEAR)BY MAKE

This example will list the make, model, modelyear of all the records in USCars by their make.

The last request is the **RETRIEVE COMMON .** It is used to merge two files by a common attribute-value. This request can be considered a transaction of two retrieve requests with a common clause in between that are processed serially in the following general template.

RETRIEVE(query-1)(target-list-1)

COMMON(attribute-1,attribute-2)

RETRIEVE(query-2)(target-list-2)

The Common attributes are numbered according to their respective RETRIEVE. For example, The following RETRIEVE-COMMON request

RETRIEVE((FILE=USCars))(MAKE,MODEL)

COMMON(MODELYEAR,MODELYEAR)

RETRIEVE((FILE=GERMANCars))(MAKE,MODEL)

will find all records in the USCars and GERMANCars with the same MODELYEAR and return a list of the MAKE and MODEL. In this case, the target lists of the two RETRIEVE requests are the same but that is not required.

These five database operations are simple, yet powerful enough to be *complete*. They are complete in the sense that typical storage, retrieval and update of data can easily be accomplished.

B. THE SOCKET INTERFACE

1. Socket Interface between MBDS and GLAD

The Socket interface was written by LT. Hogan [Ref. 11]. It required the integration of three separate programming tasks, in three different programming

environments. The first task was to implement a "server" version of the MBDS system which would allow the GLAD system to remotely interact with MBDS. The second task involved implementing a separate socket interface written in Microsoft C that the personal Computer using windows could connect to MBDS. The last task was to connect

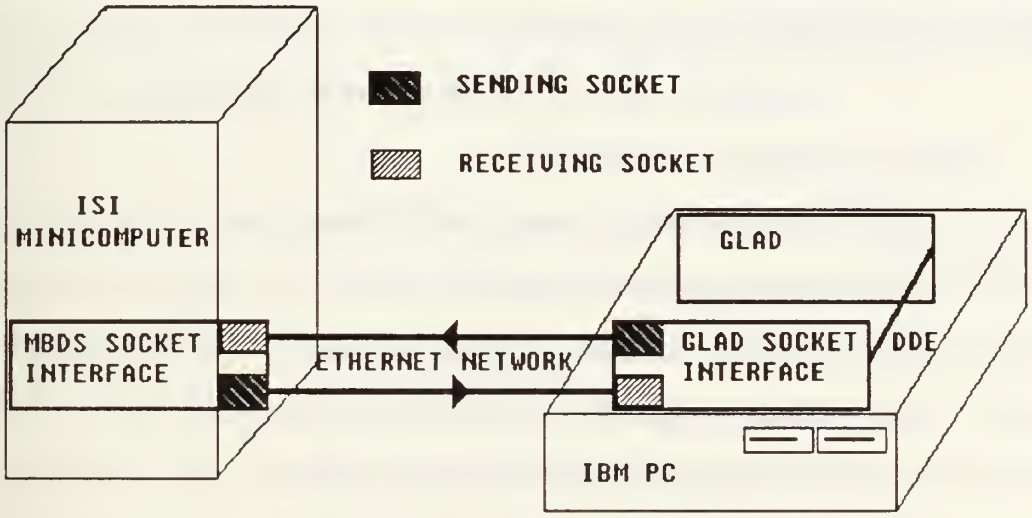

FIGURE 3.5 MBDS/GLAD Interface

the Window's socket to the Glad Environment. Figure 3.5 shows an overview of the MBDS/GLAD interface.

a. The MBDS Socket Interface

The *MBDS Socket Interface* establishes a communication link via an Ethernet network from the backend, in this case the ISI minicomputer to the front end, an IBM compatible PC. To enhance the reliability of the system and to aid in debugging, LT. Hogan chose to use separate sending and receiving sockets on both the backend and the front. As shown in Figure 3.5, all the queries from GLAD are sent via DDE, dynamic

data exchange, by messages to the GLAD socket interface, which transmits them through its sending socket to MBDS's receiving socket. On the reverse, all MBDS data and error messages are sent through MBDS's sending socket to the GLAD socket interfaces' receiving socket, and are then sent via DDE messages to GLAD.

To activate the MBDS socket interface, they created a *Test Interface, TI*, portion of MBDS. The TI is the user-interface for MBDS and presents the user with the following top-level menu upon starting MBDS.

The Multi-Lingual/Multi-Backend Database System

Select an operation:

(a) - Execute the attribute-based/ABDL interface

(r) - Execute the relational/SQL interface

(h) - Execute the hierarchical/ DL/I interface

(n) - Execute the network/CODASYL interface

(f) - Execute the functional/DAPLEX interface

(g) - Execute the object-oriented/GLAD interface

(x) - Exit to the operating system

Select->__

At the present time, a user must first log onto the ISI minicomputer and start the Multi-Lingual/Multi-Backend Database System. To initiate an interface with the GLAD system, a user selects operation (g) which sets up the MBDS socket interface. This must be done prior to attempting to open any MBDS databases from the GLAD system on the PC. A future enhancement will be able to place the MBDS in the server mode directly from within GLAD.

The receiving socket is set up first. Then the sending socket is created and the MBDS system enters its server mode, awaiting messages from GLAD. Messages are

sent over the network by sending the length of the message. Based on this message length, storage is dynamically allocated for the incoming message. The first three bytes of any message is the message type. Presently, three types of messages are supported by MBDS. The three are *open database, query database,* and *terminate session.* If an improper or invalid message is received, an appropriate error message is sent back to GLAD.

The *open database* option allows GLAD to initiate an MBDS database for use. An example of an open database message is:

010CARS

The 010 is the code used to signify that this is an "open database" message and CARS is the name of the database to be opened. After all the records have been loaded, the user and database IDs are broadcast to the 12 processes which make up the MBDS system. If MBDS is unable to open and load the database properly, an appropriate error message is sent over the network to GLAD. If MBDS is successful, and the database is loaded then MBDS waits to process other incoming messages.

The *query database* option allows the GLAD user access to the data in any previously opened MBDS database. Below is an example of a "query database" message.

020CARS@[RETRIEVE((FILE=USCars)and(MAKE=Dodge))(MODEL)]

The 020 is the code used to signify that this is a query database message. An "@" symbol is used to delimit the end of the database name, which in the above query is CARS. Anything following the "@" sign is the actual query itself. The query is sent to the *Request Preparation* which handles the parsing and execution of the query. After MBDS has processed the query, the results are gathered and translated to the GLAD's data format, which consists of an "&" character following each attribute, and carriage return and line feed (CR-LF) at the end of each record. MBDS uses an "@" symbol

rather than the CR-LF at the end of each record in order to transmit the results through the socket in one continuous stream. On the GLAD side, the GLAD's socket interface saves the results in a text file, and replaces the "@" symbols with CR-LF's. MBDS has a restriction on the length of the query result responses. If a response is too large, it is broken up into segments and an "end of results" marker is placed at the end of the final segment. These segments are then transmitted back to GLAD, followed by a special message indicating that all query results have been transmitted, and MBDS awaits the next message from GLAD.

The *Terminate Session* option is sent when GLAD wishes to close the sockets interface to MBDS. The message length of zero is transmitted. When MBDS receives this terminate session message, it leaves the server mode, closes its sockets, and returns to the top-level menu, allowing the user to restart a new session or select another MBDS option.

b. The Glad Socket Interface

GLAD uses Window's Dynamic Data Exchange protocol to sends messages with MBDS request to the socket interface. The socket interface receives these messages in the form of WM_DDE_REQUEST messages in which the string containing GLAD's request is referenced by an atom that is passed in the message. The message is then sent to MBDS, first the message length then the message itself. There is no error checking on the GLAD side, all of it is done on the MBDS side.

When MBDS returns the results from a query, they are stored in a text file named "qresults.fil" on the PC. As mentioned above, the "@" symbol at the end of each record returned by MBDS is replaced by a CR-LF combination prior to storing it in the text file. Each record is stored on a different line in the GLAD data format. When all

the results of the query are gathered in the file, a WM_DDE_DATA message is sent to indicate that the data has been received. [Ref. 12]

IV. IMPLEMENTATION

In this chapter, we will demonstrate the ability to use a Graphics Language Interface on a front-end PC and link it to a MBDS backend. We will discuss the Graphics Language for Database (GLAD), and why we used it. We will discuss some of the modifications made to the existing GLAD, and what new classes were added. Finally, we will present a sample session of GLAD with a sample database.

A. GRAPHICS LANGUAGE FOR DATABASE

1. Background

The Basis of this thesis was to explore the possibilities of providing a graphical interface on a PC to a MBDS backend. GLAD is an ongoing project developed and supervised by Professor C. Thomas Wu at the Naval Postgraduate School in Monterey, CA. The reason for developing GLAD stems from the realization that an end-user visual interaction tool was needed for the database systems. Glad will provide the end-user , regardless of the type of database system (relational, network, hierarchical), with a coherent interface. This will allow the user to visually interact with the system for data manipulation and program development. The current GLAD prototype system is implemented with data definition, data manipulation, on-line help system, and the ability to store and manipulate graphic images as a part of a database.

2. Hardware and Software requirements

In chapter two, the fundamentals of object-oriented programming and Actor were discussed. GLAD was developed using the Actor programming language under MS-Windows version 2.1 . Using Actor allowed for rapid prototyping of the GLAD

system. For most programming languages, it takes a great deal of time to learn even the basics. This is not the case of Actor, Actor allows the programmer to create simple programs within a matter of days. The complexity of the programs grows rapidly. The goal of this thesis was not to build a complete system, but a system that would demonstrate the feasibility of implementing our goals. Actor gives the programmer the flexibility to quickly test interface design alternatives and make changes to the designs rapidly. See [Ref. 12,13,14] for a more detailed discussion of the advantages of using Actor to implement GLAD.

The GLAD prototype system as it was implemented for this thesis requires the following :

- IBM compatible computer (80286 or better)
- Minimum of 640K of memory (one to four expanded recommended)
- Hard disk
- Graphics display and adapter (EGA or better recommended)
- Mouse (or other pointing device)
- Microsoft Windows version 2.03 or higher
- MS-DOS version 2.0 or higher

The network used for this thesis was the Ethernet computer network and it requires:

- Excelan EXOS 205T Model 4 Intelligent Ethernet Controller Board
- Excelan LAN WorkPlace Network Software for PC DOS TCP/IP Transport System
- Excelan LAN WorkPlace Network Software for PC DOS Socket Library Application Program Interface

The Microsoft Windows environment and that of GLAD are very CPU and Memory intensive, so the amount of random access memory, hard disk access time and the speed of the CPU determines the user response times. But for the prototype system, the Zenith 248 was sufficient for both development and implementation.

B. MODIFICATIONS

One of the great advantages in using GLAD under the Actor environment is the ease at which a programmer can modify already existing programs. This modularity allows changes to be made without having to change existing methods and in most cases, one does not even need to know anything about other methods.

1. Resource File

The only change required in the ACTOR.RC file was the insertion of the menu format for the new QueryWindow class. A complete listing of ACTOR.RC is contained in Appendix A. A sample of the ACTOR.RC that was modified is listed below.

```
QueryGlad MENU
BEGIN
 MENUITEM "&Send Query",21
 POPUP "&Edit"
     BEGIN
     MENUITEM "Cu&t\tShift+Del", EDIT_CUT
     MENUITEM "&Copy\tCtrl+Ins", EDIT_COPY
     MENUITEM "&Paste\tShift+Ins", EDIT_PASTE
     MENUITEM "C&lear\tDel", EDIT_CLEAR
     MENUITEM SEPARATOR
     MENUITEM "Select &All\tCtrl+A", EDIT_SELALL
     END
```

```
MENUITEM "&Describe",12
POPUP "Q&uery"
    BEGIN
    MENUITEM "&Insert", 22
    MENUITEM "&Update", 24
    MENUITEM "&Delete", 25
    MENUITEM SEPARATOR
    MENUITEM "&Retrieve", 23
    MENUITEM "Retrieve &Common", 26
    END
POPUP "&Templates"
    BEGIN
    MENUITEM "&Insert...", 32
    MENUITEM "&Update...", 34
    MENUITEM "&Delete...", 35
    MENUITEM SEPARATOR
    MENUITEM "&Retrieve...", 33
    MENUITEM "Retrieve &Common...", 36
    END
MENUITEM "&QUIT",11
MENUITEM  "\aF1 Help",  10,Help
END
```

This is the code required to set up the QueryWindow class menu. After all changes were made the RC file was compiled, this produced a new ACTOR.EXE file with all the changes.

2. DMWindow Class

The Data Manipulation Window gives the user the ability to manipulate the database schema and its data. Since GLAD is based on an object-relationship model, objects (entities) of the database schema are shown as rectangular boxes in the DMWindow. For this thesis, the only modifications to the DMWindow Class were the addition of two methods. They were Query and returnQuery. The window for the interface to allow queries of the MBDS is created in a class called QueryWindow. Since QueryWindow is a descendent of EditWindow class and not DMWindow class, there had to be a connection to pass information. The Query method initiates the call to QueryWindow when **Query** is selected in the DMWindow, see figure 4.1. The Query

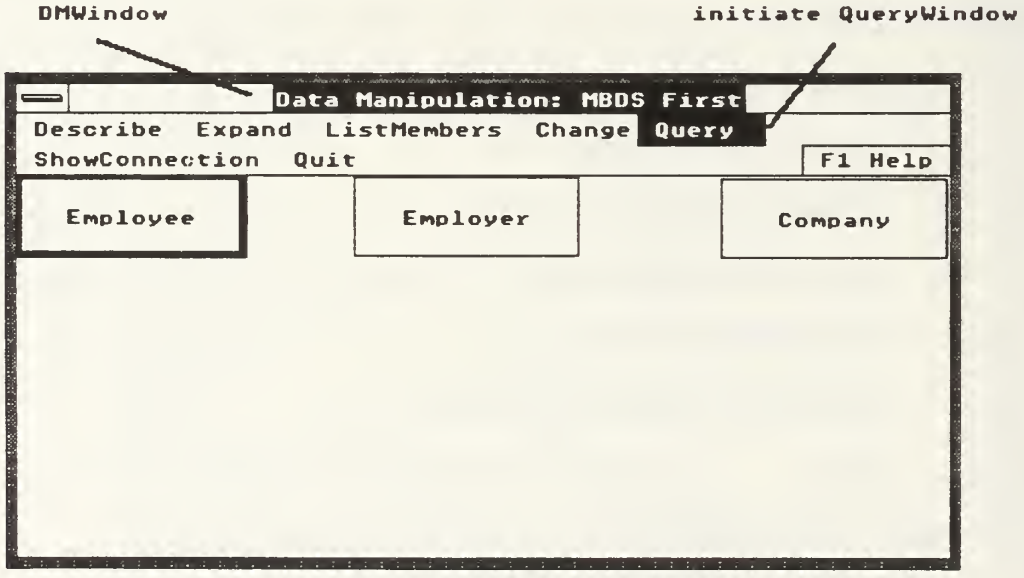

Figure 4.1 Initiate QueryWindow from DMWindow

method creates a new window called QueryWindow and passes pieces of information from DMWindow that QueryWindow will need to properly access the records of the MBDS. The returnQuery is a method that QueryWindow calls to use some of the

methods already built in DMWindow. This method is called from QueryWindow after a record has been *INSERTED, UPDATED,* or *DELETED.* This is an example of the reusability of previously written code. The code for DMWindow.cls is listed in Appendix B.

C. IMPROVEMENTS

1. QueryWindow Class

The QueryWindow class gives the user a Window in which they call, create and send Queries, see figure 4.2. At this time, it has been implemented to send queries to the backend MBDS system, in the ABDL format. After the query has been sent the results

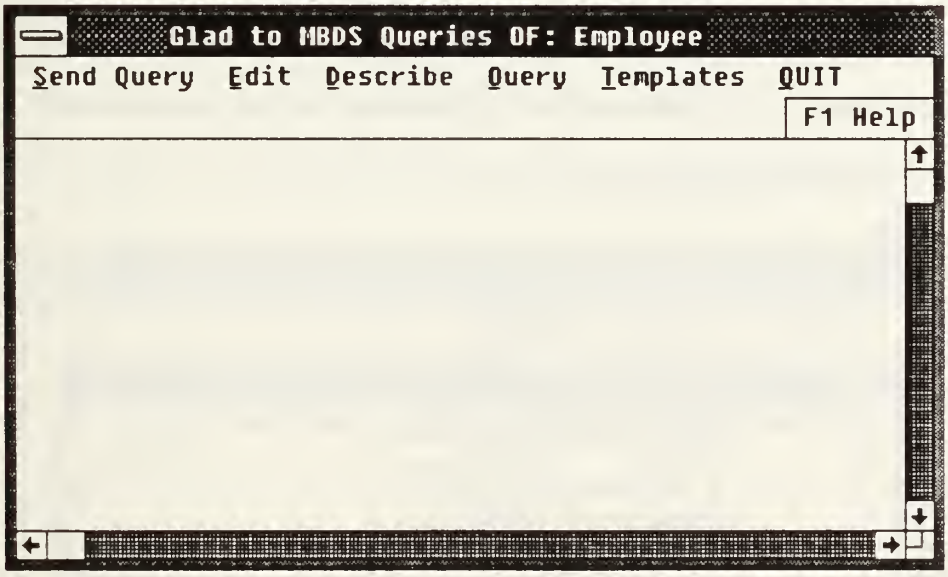

Figure 4.2 QueryWindow

are then displayed in the *Browse Window,* which will be discussed later in this chapter. One of the future improvements will be the development of a GLAD query language, and a SQL query language to the MBDS. While in the QueryWindow, a user can select the

type of query they desire by selecting the **Query** option in the QueryWindow menu. This causes a pop-up menu to appear which shows five options. The five options are the five primary operations allowed by ABDL as discussed in the previous chapter. The user can **Insert, Delete, Update, Retrieve, or Retrieve Common.** After making a selection one a user friendly template will be placed in the QueryWindow's editable area with the correct format for the query. This greatly reduces the amount an individual has to remember about the syntax and decreases the amount of input required by the user. For a more detailed template, the user can select the **Templates** option for whichever query he desires. The **Edit** option gives the user the full power of an Edit Window, to make corrections. The **Describe** option is the same as it is in the DMWindow class, it presents the attributes of the Selected Object. The **Send Query** option sends the query formed in the QueryWindow to the backend. The **Help** option gives a little explanation of how to use the QueryWindow. The next sections will discuss each of these in detail. The code for QueryWindow class is in Appendix C.

a. Edit

This menu option, when selected, gives the user a pop-up menu, see figure 4.3.

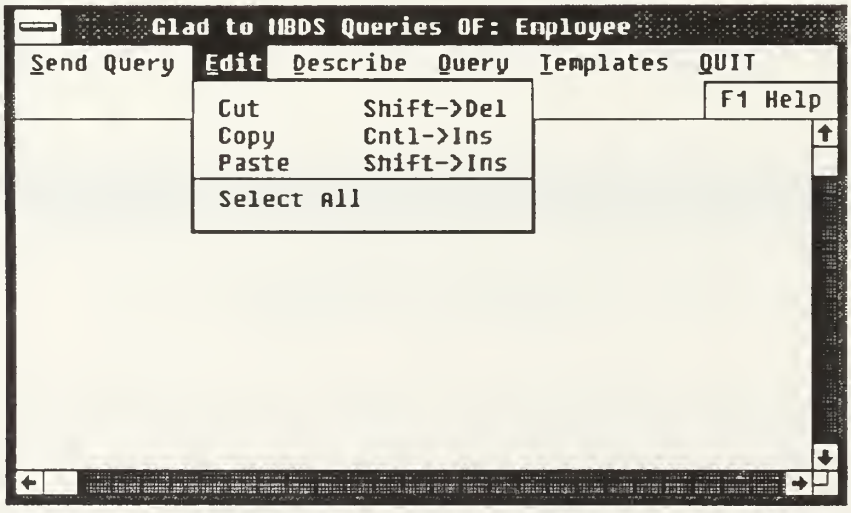

Figure 4.3 QueryWindow Edit option selected

It allows the user to manipulate the query they may have written. This allows the user to edit any errors quickly and easily. Since these are the standard edit functions used throughout Windows, little if any learning is required to use them. From the pop-up menu under **Edit,** the options available are:

- **Cut** - allows the user to remove the high-lighted text from the window and put it on the Clipboard.
- **Copy** - allows the user to make a copy of the high-lighted text from the window and it is placed on the Clipboard.
- **Paste** - allows the user to take what has been placed on the Clipboard and insert it into the QueryWindow.
- **Select All** - allows the user to high-light all the text in the QueryWindow.

b. Query

This menu option, when selected, gives the user a pop-up menu, see figure 4.4. This **Query** option gives the user five options; **Insert, Delete, Update, Retrieve, or**

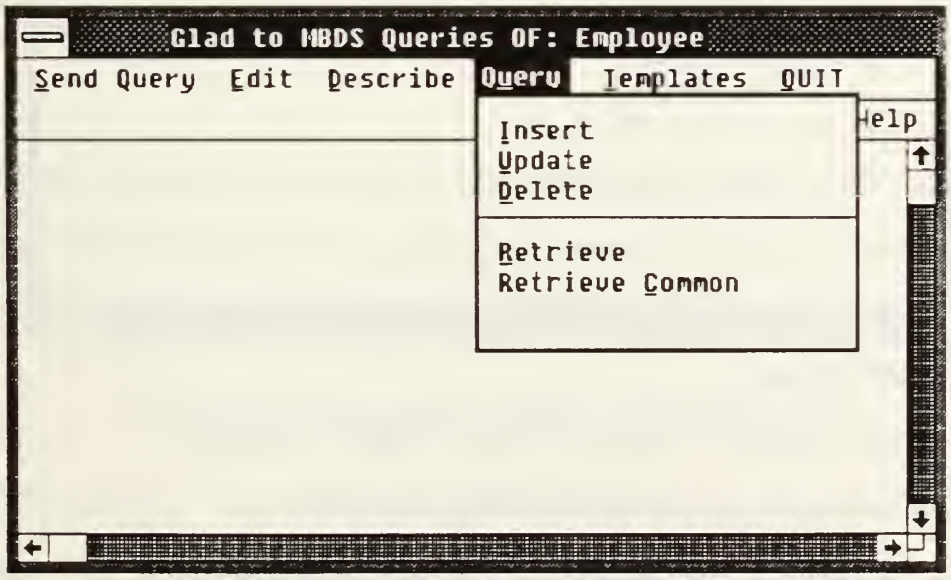

Figure 4.4 Query option pop-up menu

Retrieve Common. After selecting one a user friendly template will be placed in the QueryWindow's editable area with the correct format for the query. Figure 4.5 show a sample of the results of selecting the **Delete** option. This greatly reduces the amount an individual has to remember about the syntax and decreases the amount of input required

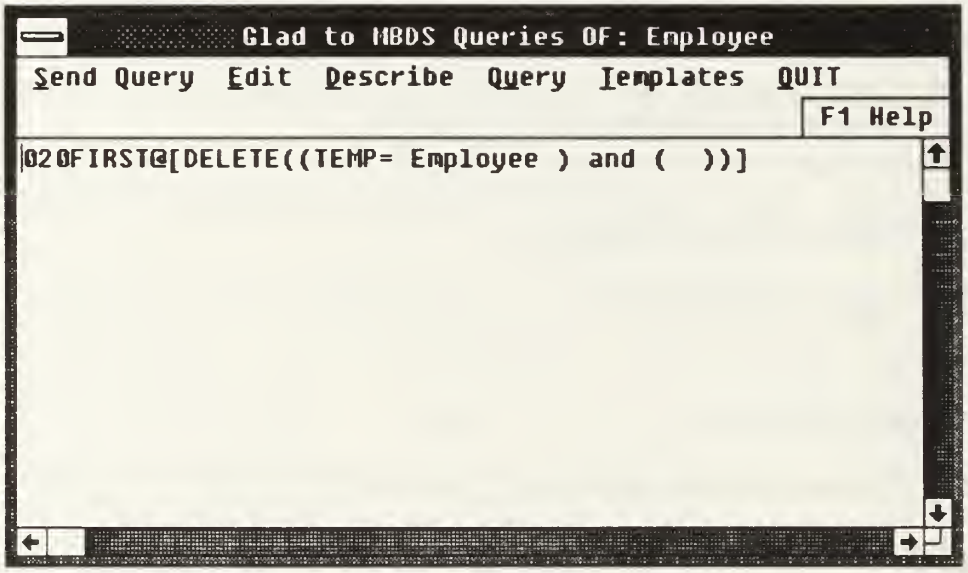

Figure 4.5 Sample QueryWindow after the Delete option of Query is selected

by the user. The *message code* 020 and the file that is open is automatically filled in as well as the *selected-object,* in this case it is Employee. The only thing that the user needs to fill in is the specifics of the *qualification* of the transactions. For example, he could type in **NAME=Hogan.**

[DELETE((TEMP=Employee) and (**NAME = Hogan**))]

All five of **Query** options give the user similar templates

- **Insert** - 020FIRST@[INSERT(<TEMP=Employee>,< >)]
- **Update** - 020FIRST@[UPDATE((TEMP=Employee)and(=)) < = >)]

- **Delete** - 020FIRST@[DELETE((TEMP=Employee) and (=))]
- **Retrieve** -020FIRST@[RETRIEVE(TEMP=Employee)(, ,)]
- **Retrieve Common** - 020FIRST@[RETRIEVE(TEMP=Employee)(, ,)]
  COMMON(,)
  [RETRIEVE(TEMP=)(, ,)]

These allow the user to do any type of data manipulation on the records from the front-end PC more easily than a user can on the backend system as they exist today.

c. Templates

Every language has constructs that control the execution of a program. The ABDL language is no different. The **Template** option , see figure 4.6, when selected will display a more detailed template for each of the five ABDL operations. These are

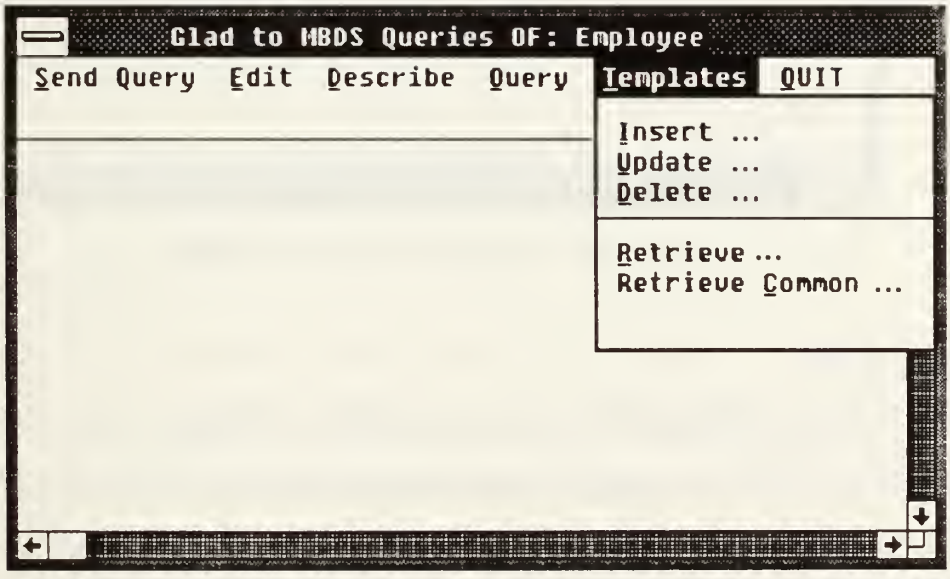

Figure 4.6 The Template options of QueryWindow

displayed in a separate window above the QueryWindow so they can be visible for the user while they type in their query. Figure 4.7 shows an example of the **Delete Template.** A further enhancement will allow the user to copy and paste directly

between the Template Window and QueryWindow. There is a template for each of the five ABDL operations.

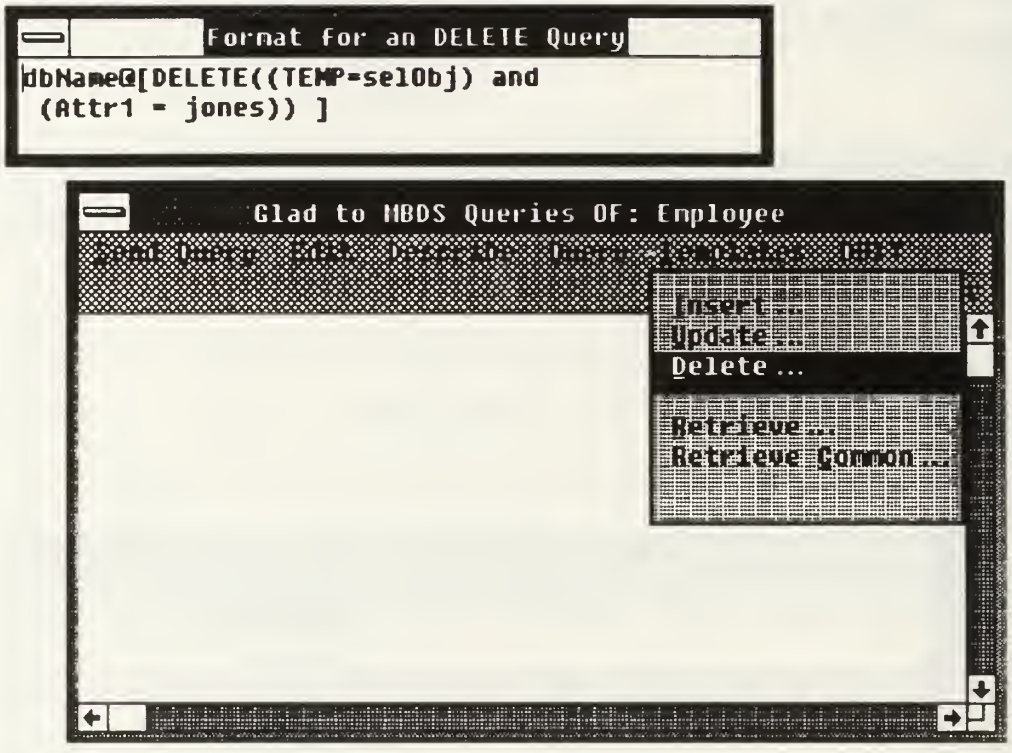

Figure 4.7 Template Window for Delete

d. Describe

The **Describe** option calls a method back in DMWindow class. This is useful to allow the user to see the structure and the attributes for the selected-object that they are presently working on. Figure 4.8 shows an example where Employee is the selected-object and the Window shows Employee's four attributes NAME, AGE, SALARY, and PHONE. This provides the user with all the information needed to formulate the query correctly. Combining the templates for each query inserted in the QueryWindow, and the more detailed templates accessed through the **Templates** option,

a user has a complete picture of what he needs for each query. All three windows remain visible as long as the user needs them.

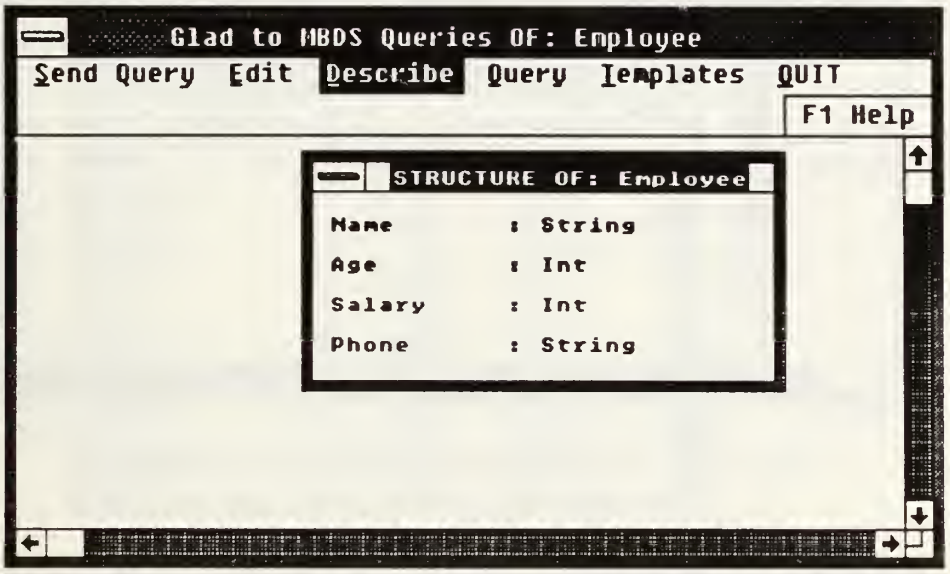

Figure 4.8 The Describe option of QueryWindow

e. Send Query

The **Send Query** option takes the query that is displayed in the QueryWindow and sends it to the backend, see figure 4.9. At this time there is no error checking accomplished in GLAD. All the error checking is done by the MBDS side; should an error occur, an error box will appear with a description of the error. A user can then just correct his query and resend the message. After the query is sent an "**Awaiting Results from MBDS ...**" dialog box will appear to let the user know that their query was sent, see figure 4.10. The results of the query will be displayed in the *Browser Window,* see figure 4.11.

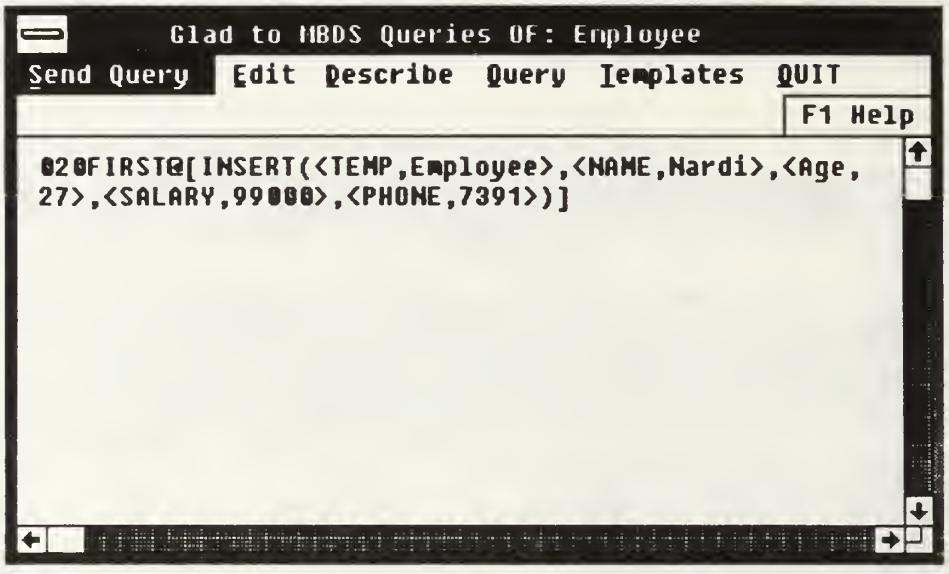

Figure 4.9 The Send Query option of QueryWindow

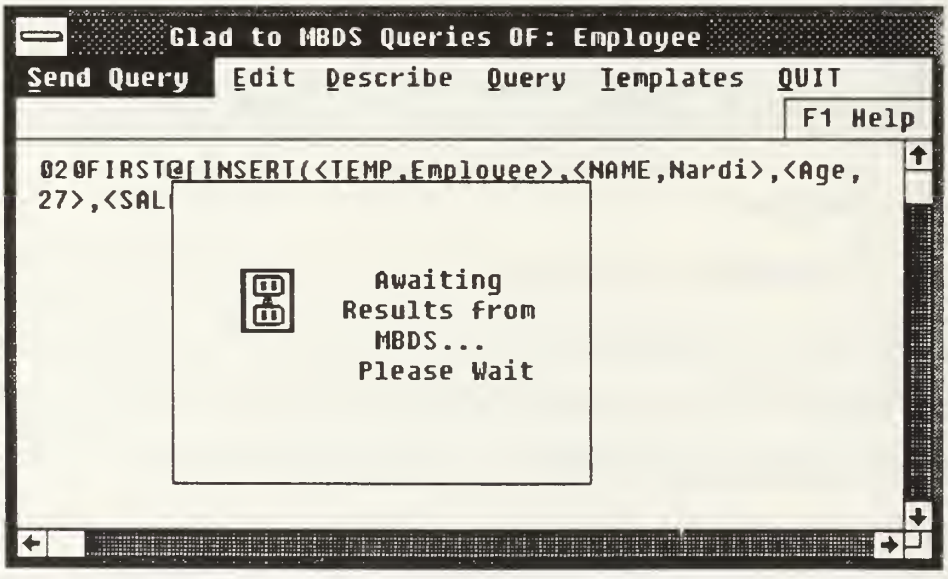

Figure 4.10 Awaiting Results from MBDS

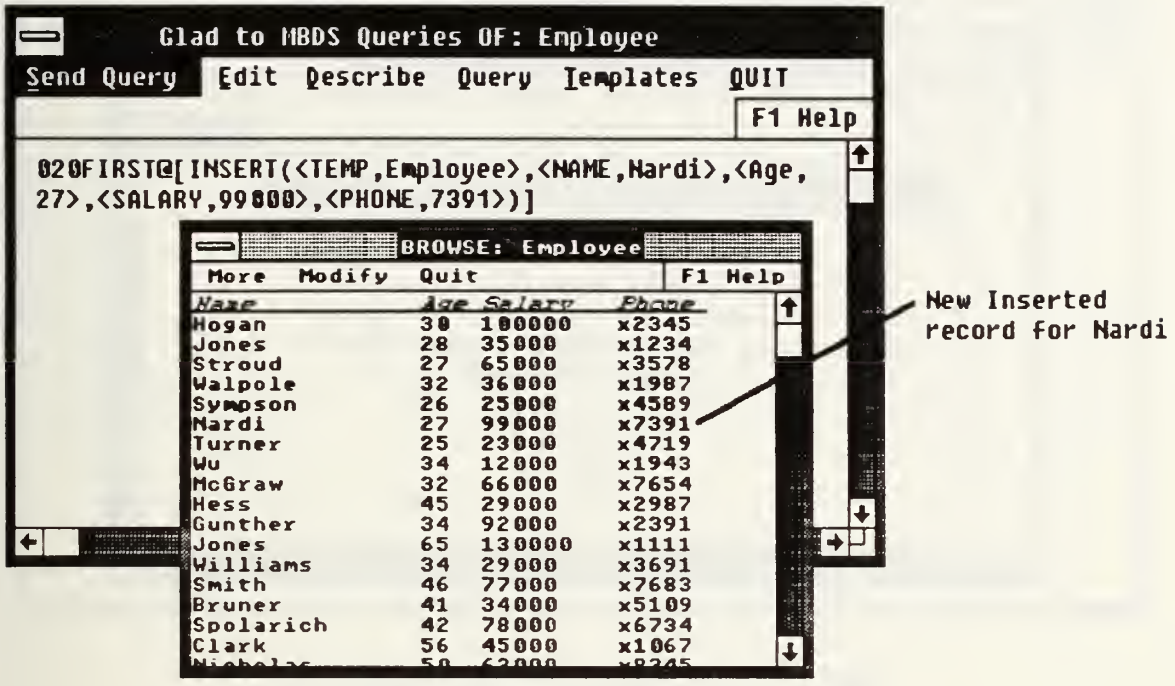

Figure 4.11 The Results displayed in the *Browse Window*

f. Help

This option at present only displays a dialog box that gives a brief explanation of the use of the QueryWindow. In the future, this will be connected to the on-line help that was done by a previous student in the future. Figure 4.12 shows the **Help** option as it exists at the present time.

g. Quit

The **Quit** option allows the user to close the QueryWindow and all the other windows that were created during its use. This option returns us back to the DMWindow where we originally called QueryWindow.

D. SAMPLE SESSION

1. Starting GLAD to receiving Results

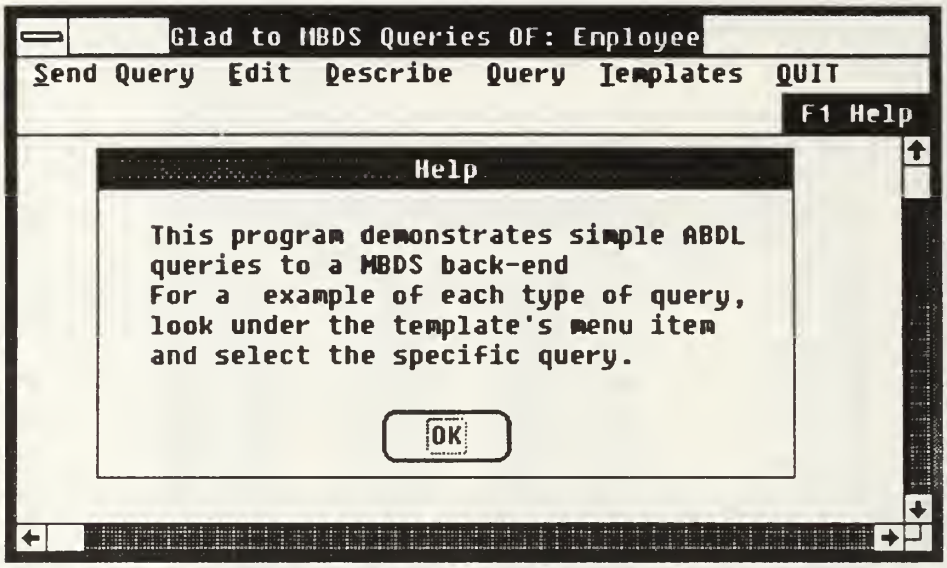

Figure 4.12 The HELP option of QueryWindow

The following is a sample session of GLAD with a sample MBDS database and the QueryWindow extension. At each level of interaction, the interface window will be described. This demonstration will enter GLAD at the beginning and walk a user through to the point where a query has been sent to MBDS and the results have returned.

a. GLAD Top-Level Window

The GLAD Top-Level Window is the starting point for all transactions. Figure 4.13 shows the GLAD Top-Level Window, from this window a user can create, modify, open or remove a database. Since GLAD was developed under the Actor environment, the use of a mouse is required for most operations. Each of the menu options available at this level will be discussed briefly.

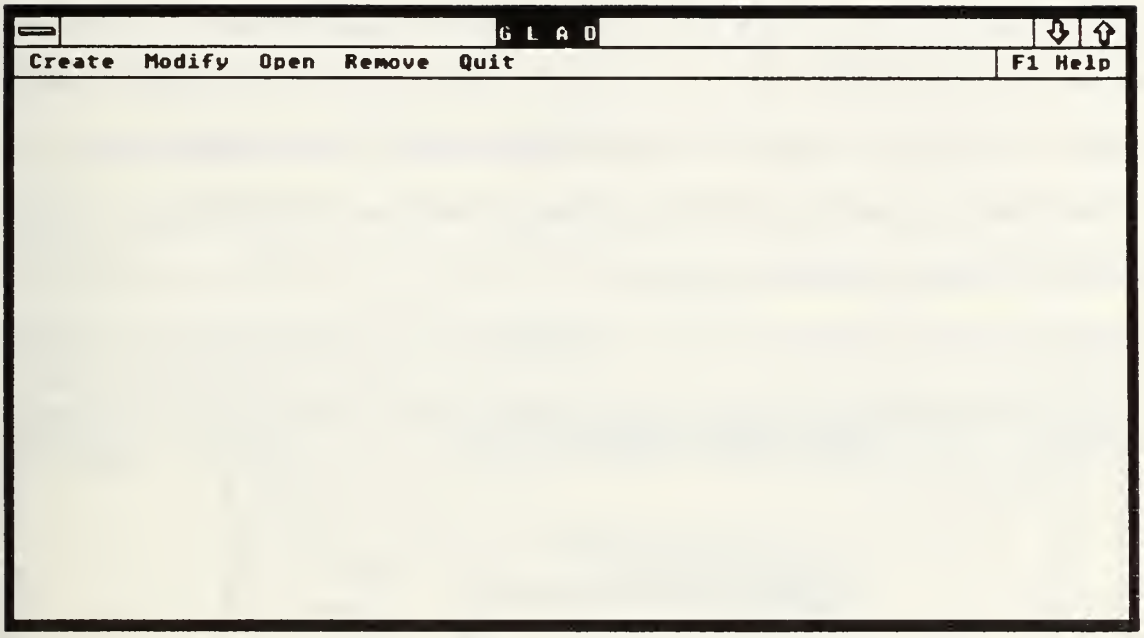

Figure 4.13 GLAD Top-Level Window

The options available under GLAD Top-Level Window are:

- **Create** - Allows the user to create a new GLAD database.
- **Modify** - Allows the user to modify an existing database schema.
- **Open** - Allows the user to open a database for display and modification.
- **Remove** - Allows the user to delete an existing database from the system.

To begin our session, we select the **Open** menu option. By placing the mouse arrow on the word **Open** and double clicking the left mouse button, a dialog box is displayed, see figure 4.14. In this dialog box is a listing of all the databases that exist in the system. Here, instead of a menu, the user's options are displayed as buttons to the right of the database listing. Should there be more databases than can fit in the window, a scroll bar allows the user to move up and down the list. To open a specific database, a user has two options. The first is to point to the desired database and, as before, double click the left

mouse button. Or the user can point at the database's name, click once, then point to the button that reads **OPEN**, and click once there. This type of feature, which allows the user to do the same operation in many different ways, is used throughout the GLAD environment. It makes it easy for the experienced user as well as the beginner to quickly navigate themselves through the system.

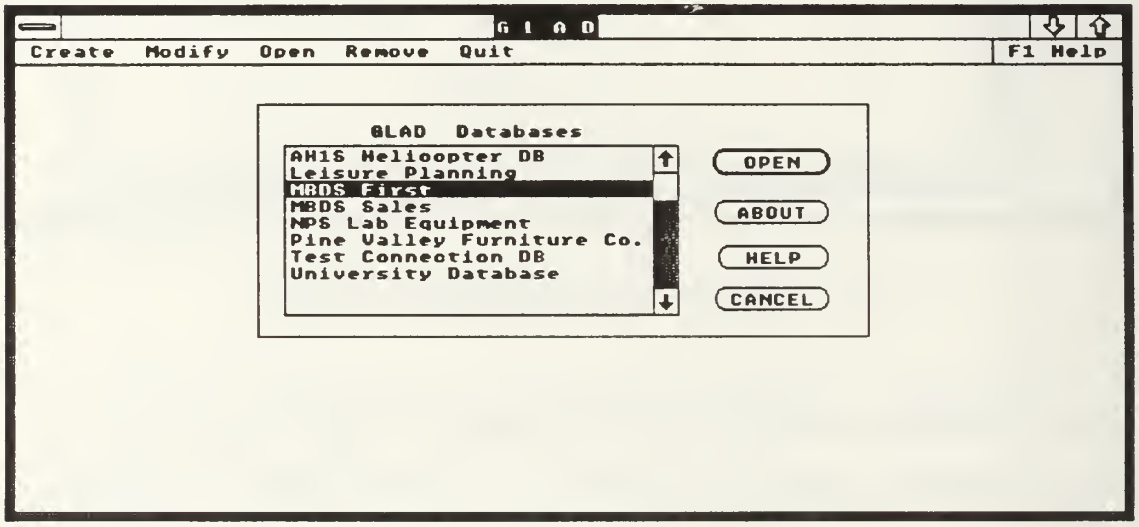

Figure 4.14 Database Selection Dialog Box

For this session we will select **MBDS First.** Notice that MBDS First is in the white letters on a black background, (this is called *high-lighted)* showing that the item has been selected. As mentioned above, to select an item point the mouse cursor on it and click the left mouse button once. All the MBDS databases have the prefix MBDS prior to their name. When a MBDS database is opened, the socket interface must be set up between GLAD and MBDS. Once both sockets are set up, GLAD sends a request to MBDS to open the database, in this case the **First** database. After MBDS acknowledges that the database is opened, then GLAD displays the data manipulation window (DMWindow) for this database using a locally stored database schema file. A further

improvement will not require the database schema to be stored locally but it would be sent from MBDS.

b. Data Manipulation Window (DMWindow)

The DMWindow displays the objects of the database, represented as rectangular boxes with the object's name in the center of each box. Figure 4.15 shows an

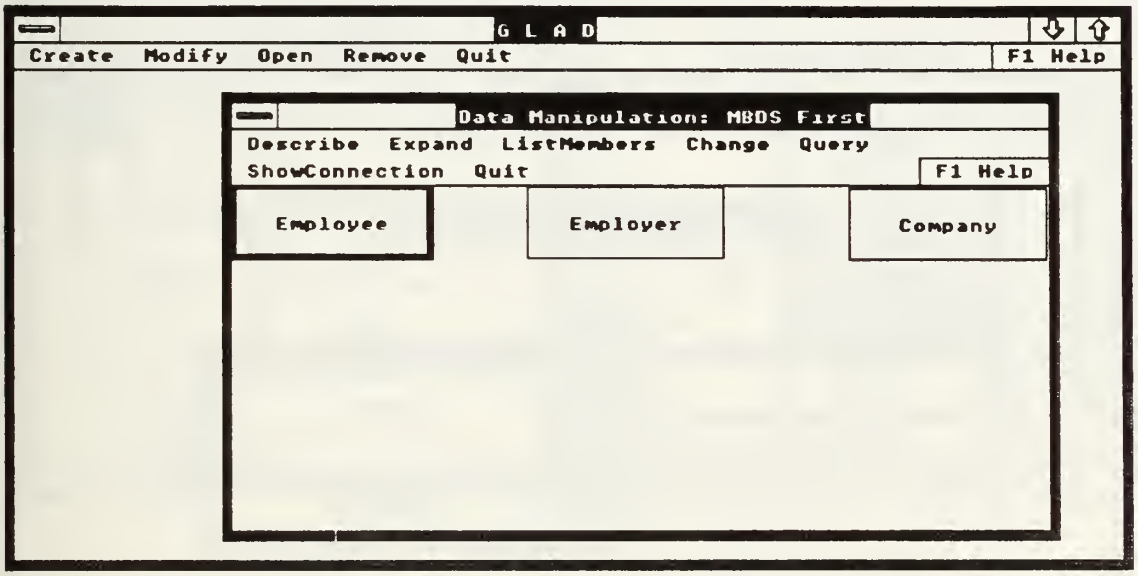

Figure 4.15 The Data Manipulation Window (DMWindow)

example of a DMWindow with our First database. The menu options for the DMWindow are:

- **Describe** - Allows the user to display the attributes of the selected objects. Figure 4.16 shows an example with Employee as the selected object.
- **Expand** - Allows the user to display sub-classes of the selected object.
- **ListMembers** - Allows the user to display and modify the object's data using an all-at-once(Browse) or one member(Display) at a time. Figure 4.17 shows an example of both. The **BROWSE** window shows all the data for Employee while the **DISPLAY** shows one record, in this case, Jones.
- **Change** - not implemented at this time.

- **Query** - Allows the user to query MBDS databases using ABDL queries. When selected it calls up the QueryWindow, see later section for more details.

- **ShowConnection** - not implemented at this time.

c. List Member Window

This window displays all the records of the database object. Here again the scroll bars enable the user to access portions of the database which do not fit in the

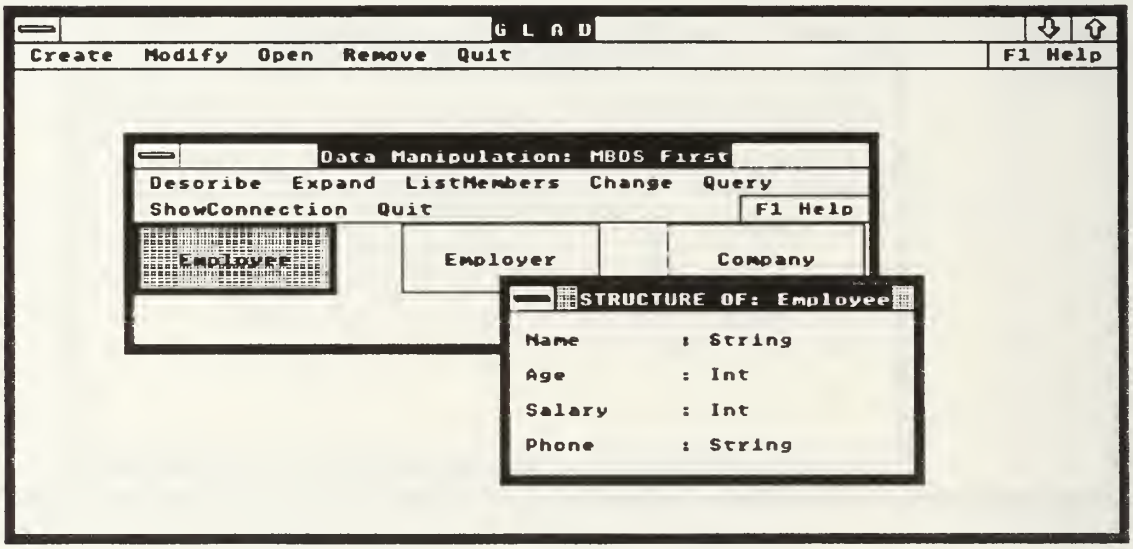

Figure 4.16 DMWindow's DESCRIBE option on Employee

window. Figure 4.17 shows the records of the Employee object. If a user wants to view an individual record in detail, he can select the **More** option after selecting that specific record. This will call the Display One Window.

d. Display One Window

Figure 4.17 also shows a Display One Window which, as described above, enables a user to see all the information contained in a specific record.

In the Display One Window, the user has the following options:

- **Add** - add a new record to the selected object.

- **Delete** - remove this selected record.
- **Modify** - change the data in this selected record.

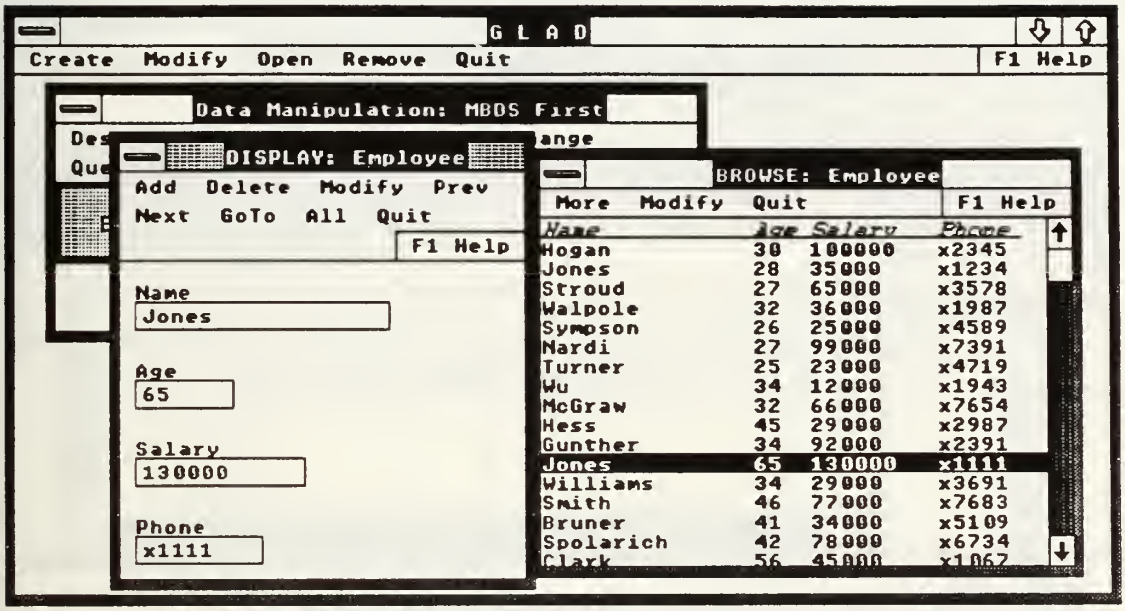

Figure 4.17 GLAD Browse and Display Windows

- **Prev** - move one record up in the list.
- **Next** - move one record down in the list.
- **Goto** - Allows the user to move to the first, last, or Ith record.
- **All** - Opens a **BROWSE** or List Member Window.

e. QueryWindow

If the user desires to query the MBDS database, they would select the **Query** option in the data manipulation window menu. Figure 4.18 shows the selection and the results of the selection. After selecting **Query,** the query window is displayed. For a more detailed discussion on each of the menu options, please refer to the previous section on QueryWindow. For this session, we will skip over some of the details. The next step

is to create our query. What we want to do is insert a new record for Nardi in the selected object, Employee. After selecting the **Query** option of QueryWindow, we fill in the details of the record for Nardi, NAME, AGE, SALARY, and PHONE. If we had

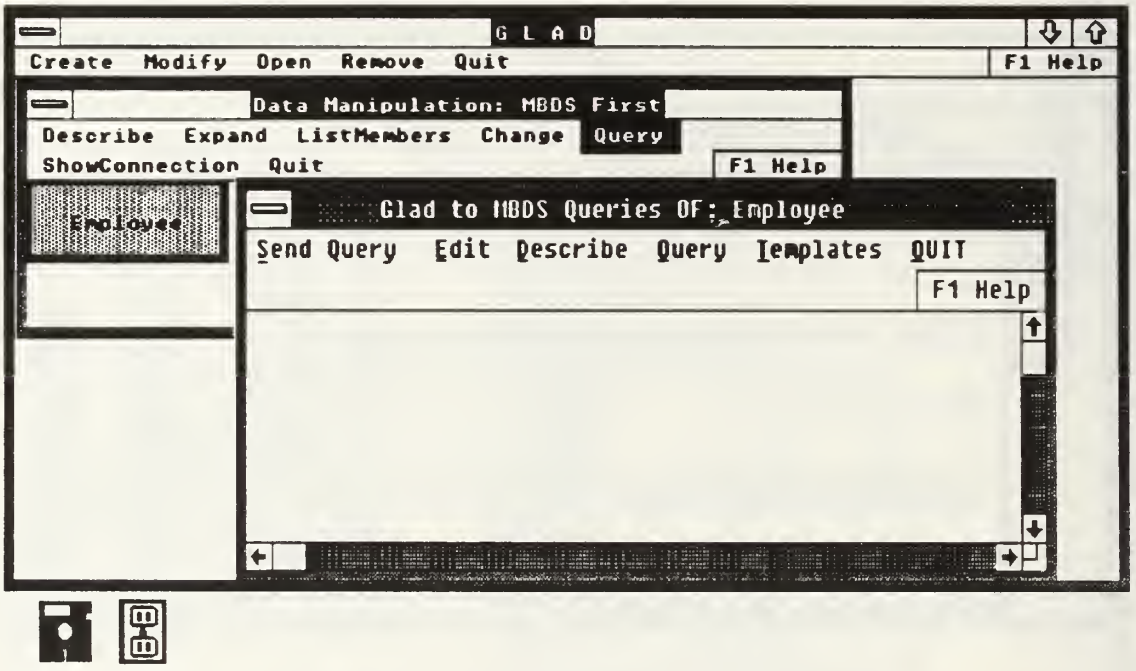

Figure 4.18 Initiating a Query from DMWindow

forgotten the attributes of Employee, we could select the Describe option and they would be displayed exactly as they are displayed in the DMWindow. If by chance, a more detailed template for the insert query is needed, we could select **Insert** from the pop-up menu of the **Templates** option of QueryWindow. Now that we have the query correct, see figure 4.19, we are ready to send it to the backend, MBDS. To do this we select the option **Send Query**, notice that it is high-lighted. After we select Send Query, the query is sent to the MBDS via the socket interfaces. Figure 4.20 shows the message that the query has been sent to MBDS and that GLAD is waiting for the results. Should the query have an error in it, a message from MBDS will tell the user the type of error. The user can then edit the query in the QueryWindow to correct any mistakes and then resend

the query. When MBDS has completed the query, it will send the results back to GLAD where they will be displayed for the user. Figure 4.21 shows the results of our insertion

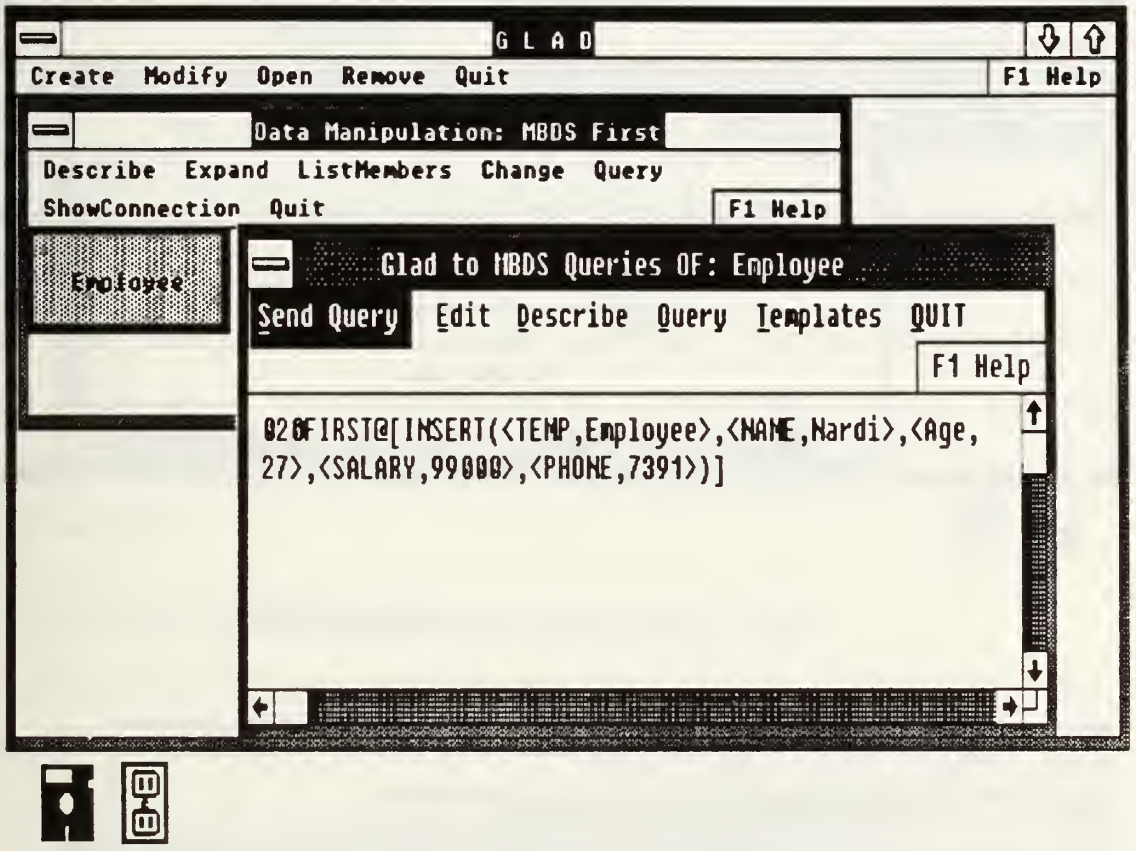

Figure 4.19 Query ready to be sent to MBDS from QueryWindow

of Nardi into the Employee record. The Browse window is automatically presented for Insert, Delete, and Update for the user's convenience.

This interface is a vast improvement over the present MBDS interface. One of the ways to further improve the GLAD interface is to allow the user to insert, delete, and update from the Display One Window. This is not implemented at the present time.

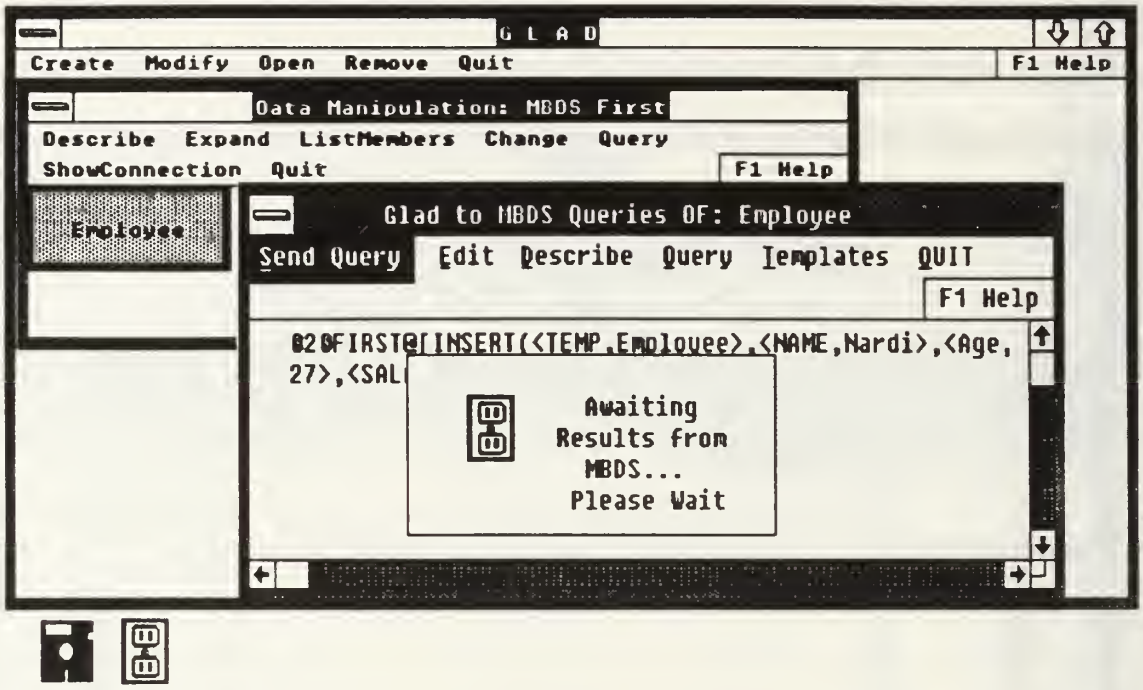

Figure 4.20 Awaiting Results from MBDS

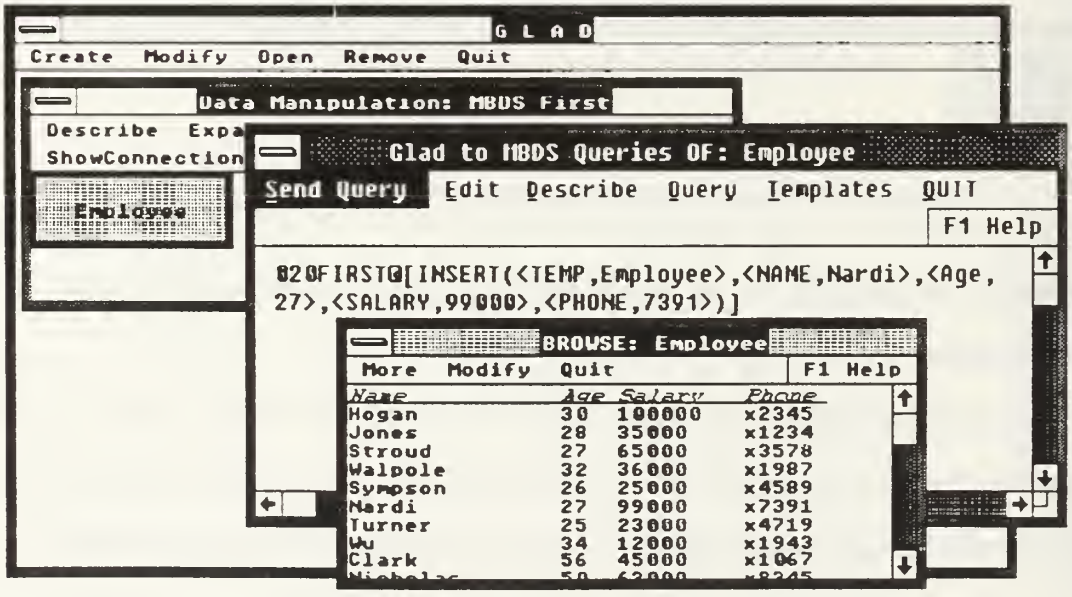

Figure 4.21 The Results of the Query displayed in the BROWSE Window

V. CONCLUSIONS

A. A REVIEW OF THE RESEARCH

The goal of the research conducted in this thesis was to provide an extension to GLAD that would provide the user with an ABDL Query ability to access the database on the MBDS backend. The graphical interface provides an user friendly system that any user at any level of experience could use immediately. The QueryWindow allows the user to use any of the five ABDL queries. This window also with some minor modifications will allow any type of queries, SQL or GLAD's own query language when it is developed.

B. FUTURE ENHANCEMENTS

Since the GLAD system is still evolving, there are many possible enhancements to the present system which include, but are certainly not limited, to the following:

- **Improved Socket Interface** - While a socket interface presently exists, there is room for improvement. We would like the socket interface to be an integral part of the GLAD system instead of its present configuration. Another improvement is to allow the user to remotely initiate the MBDS system from the individual front-ends. Also needed is the ability for the MBDS system once opened from GLAD, to send the Database's schema to the front-end instead of having the front-end constantly storing them.

- **Expand the Query Language Ability** - With the Navy's recent push towards the "paperless ship" concept, the need for the ability to generate and process the multitude of Database Models required to maintain a large navy becomes imperative. First there is a need for a GLAD query language for GLAD defined databases. Secondly, the development of a SQL query ability.

- **Digital Manuals** - One of the major goals of the "paperless ship" is to take advantage of computers' ability to store, access and assimilate data. An addition to GLAD should be a hypertext ability. This would greatly reduce the amount, weight and storage area required by the vast number of technical

and maintenance manuals required by each ship. It would also decrease the man-hours required to access the necessary data.

- **Integrated Package** - The Navy needs to develop under a user friendly interface system which every user can use. GLAD and MS-Windows provide such an environment. The package should handle word processing, AMI, databases, GLAD, and graphics, PCPaint.

- **SNAP II Backend Prototype** - The present system shows the ability and beauty of GLAD as a front-end interface for a backend. Since GLAD was developed on the Zenith 248 computer, it can be used in the Fleet today not ten years from now. The next goal should be linking GLAD to a SNAP II mini-computer and develop querying ability to the data already out in the fleet.

C. DISCUSSION AND BENEFITS OF THE RESEARCH

GLAD offers user friendly environment for the novice as well as the experienced user. The Navy has thousands of Zenith 248s in service, and a user group whose expertise varies from one end of the spectrum to the other. The vast majority have had no background what-so-ever in computers. It takes a considerable amount of time to train these individuals to perform specific tasks. Right now they are required to learn a different system for each major operation, one for databases, one for word processing and another for supply requisitions.

The GLAD and MS-Window environment gives the user one over all system to learn with many major features carrying over from one application to another. This integrated package has great potential for both the military and civilian sector. It is just such a package that has led to the popularity of the Macintosh computers by Apple. The IBM computers have as much or more power, and the software is as good as the Macintosh but Macintosh has it all integrated for the user! Replacing all the Zenith 248s and other computers with Macintosh is not the answer.

Prof. Wu has a sister project, ARGOS, being developed on the Macintosh. It is an outstanding demonstration of the power of a hypertext feature on a PC. Hypertext on IBM computers is just beginning to match the ease and power of the Macintosh.

Although this sister project named ARGOS is impressive, it unfortunately relies on Macintosh computers which are not readily available to the fleet today. GLAD is an attempt to mirror the progress made in the ARGOS project; and in area of database needs we believe we have surpasses ARGOS. GLAD is the solution today and tomorrow.

APPENDIX A. ACTOR .RC RESOURCE FILE

The following listings are the Actor code that was either created or modified for the implementation of this thesis.

```
; Resource Script File for Actor version 1.2

#include "STYLE.h"
#include "actor.h"
#include "track.h"
#include "demos.h"
#include "glad.h"


work            ICON  work.ico
Browser         ICON  browser.ico
FileWindow      ICON  filewind.ico
Inspector  ICON  inspect.ico
cube            DATA cube.dat
Actor           BITMAP      actor.bmp

gladicon ICON glad.ico
socketicon ICON  sockets.ico

GladTopMenu  MENU
BEGIN
 MENUITEM  "Create", 1
 MENUITEM  "Modify", 2
 MENUITEM  "Open",   3
 MENUITEM  "Remove", 4
 MENUITEM  "Quit",   6
 MENUITEM  "\aF1 Help",  5, HELP
END

GladTopMenu ACCELERATORS
BEGIN
 VK_F1, 5, VIRTKEY
 VK_DELETE, EDIT_CLEAR, VIRTKEY
 VK_DELETE, EDIT_CUT, VIRTKEY, SHIFT
 VK_INSERT, EDIT_COPY, VIRTKEY, CONTROL
```

```
 VK_INSERT, EDIT_PASTE, VIRTKEY, SHIFT


END

QueryGlad MENU
BEGIN
 MENUITEM "&Send Query",21
      POPUP "&Edit"
      BEGIN
      MENUITEM "Cu&t\tShift+Del", EDIT_CUT
      MENUITEM "&Copy\tCtrl+Ins", EDIT_COPY
      MENUITEM "&Paste\tShift+Ins", EDIT_PASTE
      MENUITEM "C&lear\tDel", EDIT_CLEAR
 MENUITEM SEPARATOR
      MENUITEM "Select &All\tCtrl+A", EDIT_SELALL
      END

 MENUITEM "&Describe",12
 POPUP "Q&uery"
      BEGIN
      MENUITEM "&Insert", 22
      MENUITEM "&Update", 24
      MENUITEM "&Delete", 25
      MENUITEM SEPARATOR
      MENUITEM "&Retrieve", 23
      MENUITEM "Retrieve &Common", 26
      END

POPUP "&Templates"
    BEGIN
      MENUITEM "&Insert...", 32
      MENUITEM "&Update...", 34
      MENUITEM "&Delete...", 35
      MENUITEM SEPARATOR
      MENUITEM "&Retrieve...", 33
      MENUITEM "Retrieve &Common...", 36
    END

MENUITEM "&QUIT",11
    MENUITEM  "\aF1 Help",  10,Help

END
```

```
QueryGlad ACCELERATORS
BEGIN
  VK_F1, 10, VIRTKEY

  VK_INSERT, EDIT_PASTE, VIRTKEY
  VK_DELETE, EDIT_CUT, VIRTKEY
  VK_SUBTRACT, EDIT_CUT, VIRTKEY
  VK_ADD, EDIT_COPY, VIRTKEY

  VK_LEFT, VK_LEFT, VIRTKEY
  VK_UP, VK_UP, VIRTKEY
  VK_RIGHT, VK_RIGHT, VIRTKEY
  VK_DOWN, VK_DOWN, VIRTKEY

  "^a", EDIT_SELALL
  "^s", 21
  "^z", BR_ZOOM

  VK_TAB, EDIT_TAB, VIRTKEY
  VK_PRIOR, EDIT_PRIOR, VIRTKEY
  VK_NEXT, EDIT_NEXT, VIRTKEY
  VK_HOME, EDIT_HOME, VIRTKEY
  VK_END, EDIT_END, VIRTKEY

  VK_DELETE, EDIT_CUT, VIRTKEY, SHIFT
  VK_INSERT, EDIT_COPY, VIRTKEY, CONTROL
  VK_INSERT, EDIT_PASTE, VIRTKEY, SHIFT
END



GladDmlMenu  MENU
BEGIN
  MENUITEM  "Describe",  1
  MENUITEM  "Expand",  2
  POPUP "ListMembers"
  BEGIN
 MENUITEM "All at Once",  3
 MENUITEM "One by One",  4
  END
  POPUP "Change"
  BEGIN
 MENUITEM "Add data",  5
 MENUITEM "Delete data",6
```

```
 MENUITEM "Modify data", 7
 END
 MENUITEM "Query",     8
 MENUITEM "ShowConnection", 9
 MENUITEM "Quit",      11
 MENUITEM "\aF1 Help", 10,HELP
END


GladDdlMenu MENU
BEGIN
 MENUITEM "Save",   1
 MENUITEM "Define", 2
 MENUITEM "Attribute", 3
 MENUITEM "Expand", 4
 MENUITEM "Delete", 5
 MENUITEM "Quit",   7
 MENUITEM "\aF1 Help",  6,HELP
END

GladLMMenu MENU
BEGIN
 MENUITEM "More",   1
 MENUITEM "Modify", 2
 MENUITEM "Quit",   4
 MENUITEM "\aF1 Help", 3,HELP
END


GladOMMenu MENU
BEGIN
 MENUITEM "Add",    1
 MENUITEM "Delete", 2
 MENUITEM "Modify", 3
 MENUITEM "Prev",   4
 MENUITEM "Next",   5
 POPUP "GoTo"
 BEGIN
 MENUITEM  "First", 6
 MENUITEM  "Last",  7
  MENUITEM  "I th",  8
 END
 MENUITEM "All", 9
```

```
    MENUITEM  "Quit", 11
    MENUITEM  "\aF1 Help", 10,HELP
   END



   GladCOMenu MENU
   BEGIN
    MENUITEM "Quit", 1
   END


   ABOUT_GLAD DIALOG 90,34,122,80
   STYLE WS_DLGFRAME | WS_POPUP
   BEGIN
    CTEXT  "GLAD Version 0.03",  -1, 23,12,72,11, WS_CHILD
    CTEXT  "Naval Postgraduate School", -1, 8,25,105,10,WS_CHILD
    CTEXT  "Dept of Computer Science", -1, 9,37,100,11, WS_CHILD
    ICON  "gladicon",-1,26,50,16,16, WS_CHILD
    DEFPUSHBUTTON "START", IDOK, 70,58,39,14, WS_CHILD
   END

   DATAWAIT DIALOG LOADONCALL MOVEABLE DISCARDABLE 12, 18, 98,
74
   STYLE WS_DLGFRAME | WS_POPUP
   BEGIN
         CONTROL "socketicon", -1, "static", SS_ICON | WS_CHILD, 13, 20, 16, 26
           CONTROL "Awaiting Results from MBDS...  Please Wait", 101, "static",
SS_CENTER | WS_CHILD, 36, 19, 58, 33
   END

   OPNDBLIST DIALOG LOADONCALL MOVEABLE DISCARDABLE 70, 23,
166, 102
   CAPTION "GLAD  Databases"
   STYLE WS_DLGFRAME | WS_POPUP
   BEGIN
        CONTROL  ""  DB_LB,  "listbox",  LBS_NOTIFY  |  LBS_SORT  |
LBS_STANDARD |
   WS_BORDER | WS_VSCROLL | WS_CHILD, 5, 16, 110, 82
       CONTROL "OPEN" DEFBUTTON, "button", BS_DEFPUSHBUTTON |
WS_TABSTOP |
   WS_CHILD, 125, 17, 33, 13
        CONTROL  "ABOUT"  ABOUT_DB,  "button",  BS_PUSHBUTTON  |
WS_TABSTOP |
```

```
    WS_CHILD, 125, 41, 33, 13
      CONTROL "HELP" HELP_LB, "button", BS_PUSHBUTTON | WS_TABSTOP |
WS_CHILD,
    126, 62, 32, 13
       CONTROL "CANCEL" 2, "button", BS_PUSHBUTTON | WS_TABSTOP |
WS_CHILD, 125, 82,
    33, 13
      CONTROL "GLAD  Databases" -1, "static", SS_CENTER | WS_CHILD, 17, 4,
83, 10
    END


    RMVDBLIST DIALOG LOADONCALL MOVEABLE DISCARDABLE 70, 23,
166, 102
    CAPTION "GLAD  Databases"
    STYLE WS_DLGFRAME | WS_POPUP
    BEGIN
          CONTROL "" DB_LB, "listbox", LBS_NOTIFY | LBS_SORT |
LBS_STANDARD |
    WS_BORDER | WS_VSCROLL | WS_CHILD, 5, 16, 115, 82
       CONTROL "REMOVE" DEFBUTTON, "button", BS_DEFPUSHBUTTON |
WS_TABSTOP |
    WS_CHILD, 126, 16, 33, 13
       CONTROL "CANCEL" 2, "button", BS_PUSHBUTTON | WS_TABSTOP |
WS_CHILD, 126, 81,
    33, 13
          CONTROL "ABOUT" ABOUT_DB, "button", BS_PUSHBUTTON |
WS_TABSTOP |
    WS_CHILD, 126, 39, 33, 13
      CONTROL "HELP" HELP_LB, "button", BS_PUSHBUTTON | WS_TABSTOP |
WS_CHILD,
    127, 61, 32, 13
       CONTROL "SELECT the one to be REMOVED" -1, "static", SS_CENTER |
WS_CHILD, 0, 3,
    124, 10
    END

    DEFOBJ DIALOG LOADONCALL MOVEABLE DISCARDABLE 23, 21, 136, 98
    CAPTION "OBJECT DEFINITION"
    STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
    BEGIN
      CONTROL "Enter Object Name:" 0, "static", SS_LEFT | WS_CHILD, 8, 5, 74, 10
       CONTROL "" OBJ_NAME, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP |
WS_CHILD,
```

```
    8, 16, 117, 12
        CONTROL "Atomic" ATOMIC, "button", BS_RADIOBUTTON | WS_GROUP |
WS_TABSTOP
    | WS_CHILD, 25, 44, 40, 12
        CONTROL "Nested" NESTED, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD,
    70, 44, 41, 12
            CONTROL "Nesting Level" LEVEL, "button", BS_GROUPBOX |
WS_TABSTOP |
    WS_CHILD, 20, 31, 93, 30
         CONTROL "Accept" IDOK, "button", BS_PUSHBUTTON | WS_GROUP |
WS_TABSTOP |
    WS_CHILD, 17, 70, 42, 14
            CONTROL "Cancel" IDCANCEL, "button", BS_PUSHBUTTON |
WS_TABSTOP |
    WS_CHILD, 76, 71, 42, 14
    END

    ATTRIB DIALOG LOADONCALL MOVEABLE DISCARDABLE 11, 18, 208,
216
    STYLE WS_DLGFRAME | WS_POPUP
    BEGIN
                    CONTROL "Attribute Name:" DT_CENTER, "static", SS_LEFT |
WS_CHILD, 6, 18, 64, 12
            CONTROL "Attribute Type:" 5, "static", SS_LEFT | WS_CHILD, 6, 54, 79,
12
           CONTROL "Length of field:" 15, "static", SS_LEFT | WS_CHILD, 6, 90, 86,
10
                  CONTROL "" ATTR_NAME, "edit", ES_LEFT | WS_BORDER |
WS_TABSTOP |
    WS_CHILD, 5, 30, 105, 15
                   CONTROL "" ATTR_TYPE, "edit", ES_LEFT | WS_BORDER |
WS_TABSTOP |
    WS_CHILD, 5, 67, 105, 15
                 CONTROL "" ATTR_LENGTH, "edit", ES_LEFT | WS_BORDER |
WS_TABSTOP |
    WS_CHILD, 5, 102, 105, 16
                CONTROL "" ATTR_LIST, "listbox", LBS_NOTIFY | LBS_SORT |
LBS_STANDARD |
    WS_BORDER | WS_VSCROLL | WS_CHILD, 5, 126, 106, 82
            CONTROL "Add" IDOK, "button", BS_PUSHBUTTON | WS_TABSTOP |
WS_CHILD, 138,
    42, 44, 14
```

```
            CONTROL "Delete" ATTR_DELETE, "button", BS_PUSHBUTTON |
WS_TABSTOP |
    WS_CHILD, 138, 72, 44, 14
            CONTROL "Type List" TYPE_LIST, "button", BS_PUSHBUTTON |
WS_TABSTOP |
    WS_CHILD, 138, 102, 44, 15
                CONTROL "Quit" IDCANCEL, "button", BS_PUSHBUTTON |
WS_TABSTOP |
    WS_CHILD, 139, 132, 44, 14
          CONTROL "Attributes for object" 16, "static", SS_LEFT | WS_CHILD, 29,
5, 86, 8
              CONTROL "" OBJ_NAME, "edit", ES_LEFT | WS_TABSTOP |
WS_CHILD, 118, 5, 74, 12
    END

    ATTRLIST DIALOG LOADONCALL MOVEABLE DISCARDABLE 11, 18, 122,
122
    STYLE WS_DLGFRAME | WS_POPUP
    BEGIN
          CONTROL "" TYPE_LIST, "listbox", LBS_NOTIFY | LBS_SORT |
LBS_STANDARD |
    WS_BORDER | WS_VSCROLL | WS_CHILD, 7, 6, 105, 74
       CONTROL "Accept" IDOK, "button", BS_PUSHBUTTON | WS_TABSTOP |
WS_CHILD, 18,
    90, 33, 14
           CONTROL "Cancel" IDCANCEL, "button", BS_PUSHBUTTON |
WS_TABSTOP |
    WS_CHILD, 68, 90, 29, 14
    END



    STRINGTABLE
    BEGIN
        IDSNAME, "Actor"
        IDSAPP, "ACTOR.IMA"

        1 , "Divide by 0"
        2, "Index is out of bounds"
        5, "Non-integer index argument to primitive"
        7 , "invalid size sent to new primitive"
        10, "Out of static memory"
        16, "Wrong number of block arguments"
        19, "Break occurred"
```

```
        20, "Too large for Char conversion"
        21, "Wrong number of arguments"
        22, "Wrong argument type to primitive"
        27, "Bad range to copyFrom primitive"
        32, "Can't convert to Windows short argument"
        33, "Long is too large for Int conversion"
        36, "Bad range input to munger primitive"
        40, "Primitive receiver is nil"

        syntaxError, "<<< Syntax error"
        eosError, "<<< Premature end of input"
        sLitError, "Unterminated String literal"
        undefError, "<<< Undefined variable name"
        litSymError, "<<< Incoreect literal symbol format"
        curClassError, "No current class in Compiler"
        ancestError, " is not an ancestor of "
        inheritError, " is not a function in "

        litNumError, "<<< Improper literal number format"
        wNameError, "<<< No such MS-Windows routine"
        wSynError, "<<< Improper MS-Windows call syntax"
        litArrayError, "<<< Improper literal array syntax"
        litArrayOvflError, "<<< Literal array is too large"
        defineError, "<<< Improper #define format"
        litRectError, "<<< Improper literal rectangle format"
        infixError, "<<< Not a valid infix expression"
        commentError, "<<< Unterminated comment"
        registerError, "Couldn't register class"
        menuError, "Couldn't load menu"
        wCreateError, "Couldn't create window"
        emptyError, "Empty collection"
        elemNotFndError, "Element not found in collection"
        dosError, " reported DOS error# "
        rangeError, "Index is out of bounds"
        undefCharError, "<<< Undefined"
        ivarsError, "Structs can't have instance variables"
        handleError, "No handle obtained for object"
wCallArgsError, "Wrong number of args in Windows Call"
numTempsError, "Total arguments and locals can't exceed 15"

; Used for results of checkError
  52, ", File not found"
  53, ", Path not found"
  54, ", No file handle available; all in use"
```

```
 55, ", Access denied"
 56, ", Invalid file handle"
 58, ", Insufficient memory"
 65, ", Invalid drive specification"

150, "Attempted to move freed object:"
151, "Adding to scavenger list:"
152, "Dynamic memory is full."
153, "Free list is corrupted."
154, "Scavenge list is full."
155, "Out of object pointers."
156, "Snapshot write failed."
157, "Snapshot load failed."
158, "Not enough memory to run Actor."
159, "Not enough dynamic for static gc."
160, "Actor Display"
161, "Requires higher static setting."
162, "Requires higher dynamic setting."
163, "Actor\xAE 1.2"
164, "Windows/Actor stack overflowed  "
165, "Windows/Actor stack underflowed "
166, "Actor stack overflowed"
167, "Corrupted object memory"
168, "Actor symbol table is full"

; Miscellaneous Actor system strings.  DO NOT MODIFY!
300, "Class Definition Error"
301, " cannot be redefined."
302, "Recompile these classes?"
303, "Delete these classes?"
304, "Class Name Error"
305, " already exists.  Use About Class dialog."
306, " exists.  Should it be overwritten?"
307, "File Conflict"
308, "File Renamed"
309, "Old work has .BAK extension."
310, " source is unavailable."
311, "Class Source File Error"
312, " file not found in "
313, "Actor Error"
314, "FileWindow is not loaded"
315, "File Editor: Untitled"
316, "File Edit Error"
317, "The file is too large."
```

```
318, "Discard changes?"
319, "Save text as:"
320, "workmenu"
321, "Actor Workspace"
322, "** Recompile classes; remove existing instances **"
323, "Untitled"
324, "browmenu"
325, "Browser"
326, "/* class comment */"
327, " class definition */ "
328, "debugmenu"
329, "Debugger: "
330, "Can't resume!"
331, "classes\\"
332, "work\\"
333, "backup\\"
334, "EditWindow"
335, "File Error"
336, "FileEditMenu"
337, "File Editor"
338, "Breakpoint"
339, " bytes reclaimed."
340, " bytes available."
341, "Syntax Error"
342, "Recursive error:"
343, "Actor Error: "
344, "Not understood:"
345, "Recursive message send failure:"
346, " doesn't understand:"
347, "Compilation Error"
348, " is undefined. Should it become a global variable?"
349, "Undefined Name"
350, "Bytes Free"
351, "Static: "
352, "  MS-Windows: "
353, "Missing BACKUP directory for source."
354, "Write the Image to this file:"
355, "Load Error"
356,  "You must assign LoadFiles before using load()."
357, "Warning!"
358, "Dynamic memory is getting low."
359, "Run Application"
360, "Application file name:"
361, "Editor: "
```

```
    362, "Inspector: "
    363, "Browser: "
    364, "workedit"
    365, "Senders"
    366, "Implementors"
    367, "References"
    368, "Global References"
    369, "inspmenu"
    370, "Do you really wish to close this window?"
    371, "MS-Windows function "
    372, " takes "
    373, " argument(s)."
    374, " (CLASS)"
    375, " "      ; can use " (OBJECT)" if you want this label for object methods
                 ; in the Browser
    376, "Stack frames above recompiled method are now invalid."
    377, "You must exit Actor before exiting Windows"
    378, "  Dynamic: "

   ; template strings
    TEMP_DO, " do(receiver, {using(elem)  });"
    TEMP_IF, " if cond then stmtList; endif;"
    TEMP_IFEL, " if cond then stmtList; else stmtList; endif;"
    TEMP_BLOCK, " {using(elem)     }"
    TEMP_CASE, " select case cond is stmtList;endCase case cond is stmtList;endCase
endSelect;"
    TEMP_LOOP, " loop while cond begin stmtList; endLoop;"
    TEMP_NMETH, "/* comment */ Def  method(self)  { }"
   END

   Actor ACCELERATORS
   BEGIN
    VK_INSERT, EDIT_PASTE, VIRTKEY
    VK_DELETE, EDIT_CUT, VIRTKEY
    VK_SUBTRACT, EDIT_CUT, VIRTKEY
    VK_ADD, EDIT_COPY, VIRTKEY

    VK_LEFT, VK_LEFT, VIRTKEY
    VK_UP, VK_UP, VIRTKEY
    VK_RIGHT, VK_RIGHT, VIRTKEY
    VK_DOWN, VK_DOWN, VIRTKEY

    "^a", EDIT_SELALL
    "^r", BR_REFORM
```

```
    "^z", BR_ZOOM

    VK_TAB, EDIT_TAB, VIRTKEY
    VK_PRIOR, EDIT_PRIOR, VIRTKEY
    VK_NEXT, EDIT_NEXT, VIRTKEY
    VK_HOME, EDIT_HOME, VIRTKEY
    VK_END, EDIT_END, VIRTKEY

    VK_DELETE, EDIT_CUT, VIRTKEY, SHIFT
    VK_INSERT, EDIT_COPY, VIRTKEY, CONTROL
    VK_INSERT, EDIT_PASTE, VIRTKEY, SHIFT
  END

  ABOUT_BOX DIALOG DISCARDABLE 59, 79, 151, 128
  STYLE WS_POPUP | WS_DLGFRAME
  BEGIN
    CTEXT "Actor\xAE 1.2" -1, 1, 12, 147, 10
    CTEXT "Copyright \xA9 1986-1988" -1, 1, 28, 147, 10
    CTEXT "The Whitewater Group, Inc." -1, 1, 39, 147, 10
    CTEXT "All rights reserved." -1, 1, 50, 147, 10
    ICON "work" 5, 24, 98, 13, 17
    ICON "browser" 6,  114, 98, 13, 17
    CTEXT "Portions Copyright \xA9 1983-1988", -1, 1, 68, 147, 10
    CTEXT "Microsoft Corporation", -1, 1, 79, 147, 10
    DEFPUSHBUTTON "&Ok" IDOK, 57, 99, 32, 14, WS_GROUP
  END

  INPUT_BOX DIALOG DISCARDABLE 77, 94, 165, 71
  STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
  BEGIN
      EDITTEXT FILE_EDIT, 10, 32, 138, 12, WS_BORDER | WS_CHILD |
WS_TABSTOP | ES_AUTOHSCROLL
    LTEXT "", INPUT_MSG, 11, 5, 143, 18, WS_CHILD
    DEFPUSHBUTTON "&Ok" IDOK, 32, 50, 32, 14,   WS_CHILD
    PUSHBUTTON "&Cancel" IDCANCEL, 99, 50, 32, 14, WS_CHILD
  END

  ERR_BOX DIALOG DISCARDABLE 48, 32, 210, 85
  STYLE WS_POPUP | WS_CAPTION
  CAPTION "Error Dialog"
  BEGIN
    DEFPUSHBUTTON "&Ok", IDOK, 172, 8, 28, 14, WS_GROUP
    PUSHBUTTON "&Debug", IDYES, 172, 28, 28, 14, WS_GROUP
    LISTBOX ERR_LB, 4, 8, 160, 70
```

```
END

DW_BOX DIALOG DISCARDABLE 27, 27, 201, 105
STYLE WS_DLGFRAME | WS_POPUP
BEGIN
  LTEXT "The text in the Browser edit window has been" 2, 10, 11, 180, 10
  LTEXT "changed.  Accept or Cut to Clipboard?" 3, 10, 24, 150, 10
  PUSHBUTTON "&Accept", DW_ACC, 10, 47, 75, 14, WS_CHILD
  PUSHBUTTON "Cut to C&lipboard", DW_CTC, 10, 74, 75, 14, WS_CHILD
  DEFPUSHBUTTON "A&bandon", DW_ABA, 110, 47, 75, 14, WS_CHILD
  PUSHBUTTON "&Cancel", IDCANCEL, 110, 74, 75, 14, WS_CHILD
END

FRACTAL_BOX DIALOG DISCARDABLE 90, 69, 160, 85
CAPTION "Fractal Controls"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
  CONTROL "Type" -1, "button", BS_GROUPBOX | WS_CHILD, 8, 9, 66, 50
      CONTROL "&Koch" ID_KOCH, "button", BS_RADIOBUTTON |
WS_TABSTOP | WS_CHILD, 12, 20, 28, 12
    CONTROL "&Square Koch" ID_SQKOCH, "button", BS_RADIOBUTTON |
WS_CHILD, 12, 31, 56, 12
      CONTROL "&Peano" ID_PEANO, "button", BS_RADIOBUTTON |
WS_CHILD, 12, 42, 33, 12
    CONTROL "Order" -1, "button", BS_GROUPBOX | WS_GROUP | WS_CHILD,
86, 9, 30, 70
    CONTROL "&1" ID_ORDER1, "button", BS_RADIOBUTTON | WS_CHILD,
94, 20, 16, 12
    CONTROL "&2" ID_ORDER2, "button", BS_RADIOBUTTON | WS_CHILD,
94, 31, 16, 12
    CONTROL "&3" ID_ORDER3, "button", BS_RADIOBUTTON | WS_TABSTOP
| WS_CHILD, 94, 43, 16, 12
    CONTROL "&4" ID_ORDER4, "button", BS_RADIOBUTTON | WS_CHILD,
94, 54, 16, 12
    CONTROL "&5" ID_ORDER5, "button", BS_RADIOBUTTON | WS_CHILD,
94, 65, 16, 12
    CONTROL "50" ID_LENGTH, "edit", ES_LEFT | WS_BORDER | WS_GROUP |
WS_TABSTOP | WS_CHILD, 40, 67, 34, 12
    CONTROL "Length: " -1, "static", SS_LEFT | WS_CHILD, 8, 69, 28, 8
    CONTROL "&OK" IDOK, "button", BS_DEFPUSHBUTTON | WS_TABSTOP |
WS_CHILD, 124, 26, 28, 14
      CONTROL "&Cancel" IDCANCEL, "button", BS_PUSHBUTTON |
WS_TABSTOP | WS_CHILD, 124, 53, 28, 14
END
```

```
DCL_BOX DIALOG DISCARDABLE 44, 25, 234, 134
STYLE WS_DLGFRAME | WS_POPUP
BEGIN
  PUSHBUTTON "&Delete Files" DCL_DEL, 38, 97, 60, 14, WS_CHILD
  PUSHBUTTON "Save &Files" DCL_SAV, 38, 113, 60, 14, WS_CHILD
  DEFPUSHBUTTON "&Snapshot" IDOK, 136, 97, 60, 14, WS_CHILD
  PUSHBUTTON "&Cancel" IDCANCEL, 136, 113, 60, 14, WS_CHILD
  LTEXT "You have modified the image.  The modified source" 9, 12, 5, 204, 10
  LTEXT "files for the following classes are located in the" 2, 12, 16, 205, 10
  LTEXT "WORK directory." 2, 12, 27, 63, 10
  LTEXT "" DCL_LIST, 11, 43, 212, 22,  WS_BORDER | WS_CHILD
  LTEXT "Before quitting, do you want to take a snapshot, or" 5, 13, 69, 208, 10
   LTEXT "save the modified source files in the WORK directory?" 8, 12, 80, 212,
10
END


FILE_BOX DIALOG DISCARDABLE 27, 23, 192, 105
STYLE WS_DLGFRAME | WS_POPUP
BEGIN
  EDITTEXT FILE_EDIT, 54, 5, 127, 12, ES_AUTOHSCROLL | WS_CHILD
     CONTROL "" FILE_LB, "listbox", LBS_STANDARD | WS_TABSTOP |
WS_CHILD, 10, 39, 99, 57
  DEFPUSHBUTTON "&Open", IDOK, 135, 47, 32, 15, WS_CHILD
  PUSHBUTTON "&Cancel", IDCANCEL, 135, 73, 32, 15, WS_CHILD
  CONTROL "File name:" 3, "static", SS_LEFT | WS_CHILD, 10, 7, 41, 11
  CONTROL "" FILE_DIR, "static", SS_LEFT | WS_CHILD, 10, 23, 176, 11
END

CLASS_BOX DIALOG  DISCARDABLE 36,48,270,160
STYLE WS_POPUP | WS_CAPTION
CAPTION "Class Definition"

BEGIN
     LTEXT   "Name:", CLASS_LNAME, 4, 2, 20, 14
     EDITTEXT CLASS_NAME, 4, 12, 100, 14
     LTEXT   "Ancestor:", CLASS_LANC, 4, 28, 40, 14
     EDITTEXT CLASS_ANCEST, 4, 38, 100, 14
     RADIOBUTTON "&Byte", CLASS_BYTE, 6, 64, 30, 14
     RADIOBUTTON "&Word", CLASS_WORD, 40, 64, 30, 14
     RADIOBUTTON "&Ptr", CLASS_PTR, 70, 64, 30, 14
     GROUPBOX "Format", CLASS_FORM, 4, 54, 100, 26
     CHECKBOX "&Indexed", CLASS_IDX, 4, 82, 40, 14
```

```
        DEFPUSHBUTTON "Accept", IDOK, 46, 86, 28, 14     ; new default for
Windows 2.0
        PUSHBUTTON "Cancel", IDCANCEL, 76, 86, 28, 14    ; no longer the default
        LTEXT  "Variables:", CLASS_LVARS, 108, 2, 50, 14
                        EDITTEXT  CLASS_VARS,  108,  12,  160,  90,
ES_AUTOVSCROLL|ES_MULTILINE|WS_VSCROLL
        LTEXT  "Comment:", CLASS_LCOM, 4, 96, 40, 14
                        EDITTEXT  CLASS_COM,  4,  106,  264,  52,
ES_AUTOVSCROLL|ES_MULTILINE|WS_VSCROLL
    END

    ; methodbr.rc for lang ext I
    ;

    MBrowMenu  MENU
    BEGIN
     MENUITEM "&Accept!", BR_ACCEPT

     POPUP "&Edit"
      BEGIN
       MENUITEM "Cu&t\tShift+Del", EDIT_CUT
       MENUITEM "&Copy\tCtrl+Ins", EDIT_COPY
       MENUITEM "&Paste\tShift+Ins", EDIT_PASTE
       MENUITEM "C&lear", EDIT_CLEAR
       MENUITEM SEPARATOR
       MENUITEM "Select &All\tCtrl+A", EDIT_SELALL
       MENUITEM "&Reformat\tCtrl+R", BR_REFORM
      END

     MENUITEM "&Doit!", INSP_DOIT
     MENUITEM "&Inspect!", INSP_ISEL

     POPUP "&Utility"
     BEGIN
      MENUITEM "&Implementors", WORK_IMP
      MENUITEM "&Senders", WORK_SYMSEND
      MENUITEM "&Global References", WORK_GLOSEND
      MENUITEM "&References", WORK_SEND
     END

     POPUP "&Templates"
      BEGIN
       MENUITEM "&do",  TEMP_DO
       MENUITEM "&if/then", TEMP_IF
```

```
    MENUITEM "if/&else", TEMP_IFEL
    MENUITEM "&block", TEMP_BLOCK
    MENUITEM "&select/case", TEMP_CASE
    MENUITEM "&loop", TEMP_LOOP
    MENUITEM SEPARATOR
    MENUITEM "&New method", TEMP_NMETH
  END
END

demoMenu MENU
BEGIN
 MENUITEM "&Clear!",        DEMO_CLEAR
 POPUP    "&Turtle"
  BEGIN
   MENUITEM "&Load Demo",  DEMO_TURTLOAD
   MENUITEM SEPARATOR
   MENUITEM "&Pattern...", DEMO_FRACTAL
  END
 MENUITEM "T&rack!",        DEMO_TRACLOAD
 MENUITEM "C&ube!",         DEMO_CUBELOAD
 MENUITEM "&Graph!",        DEMO_GRAFLOAD
 MENUITEM "&Fractal!",      DEMO_FRACLOAD
 POPUP    "&Mandelbrot"
  BEGIN
   MENUITEM "Plot&1",     DEMO_BRT1LOAD
   MENUITEM "Plot&2",     DEMO_BRT2LOAD
   MENUITEM "Plot&3",     DEMO_BRT3LOAD
  END
 MENUITEM "&Queens!",       DEMO_QUENLOAD
 MENUITEM "C&lassTree!",    DEMO_TREELOAD
 MENUITEM "&ActorLogo!",    DEMO_LOGOLOAD
END

track MENU
BEGIN
 POPUP "&Shape"
   BEGIN
    MENUITEM "&Clear"    , IDDCLEAR
    MENUITEM "&Ellipse"  , IDDELLIPSE
    MENUITEM "&Rectangle", IDDRECT
    MENUITEM "&Star"     , IDDSTAR
    MENUITEM "&Triangle" , IDDTRIANGLE
   END
END
```

```
EditMenu  MENU
BEGIN
 POPUP "&Edit"
   BEGIN
   MENUITEM "Cu&t",  EDIT_CUT
   MENUITEM "&Copy", EDIT_COPY
   MENUITEM "&Paste", EDIT_PASTE
   MENUITEM "C&lear", EDIT_CLEAR
   END
END

DebugMenu  MENU
BEGIN
  MENUITEM "&Accept!",  BR_ACCEPT
  POPUP "&Edit"
    BEGIN
      MENUITEM "Cu&t",  EDIT_CUT
      MENUITEM "&Copy", EDIT_COPY
      MENUITEM "&Paste", EDIT_PASTE
      MENUITEM "C&lear", EDIT_CLEAR
      MENUITEM SEPARATOR
      MENUITEM "Select &All", EDIT_SELALL
      MENUITEM "&Reformat", BR_REFORM
    END
  MENUITEM "&Doit!", INSP_DOIT
  POPUP "&Inspect"
    BEGIN
      MENUITEM "&Temporary", DBG_TEMP
      MENUITEM "&Selection", INSP_ISEL
    END
  POPUP "&Utility"
   BEGIN
     MENUITEM "&Implementors", WORK_IMP
     MENUITEM "&Senders", WORK_SYMSEND
     MENUITEM "&Global References", WORK_GLOSEND
     MENUITEM "&References", WORK_SEND
   END
  MENUITEM "&Resume!", DBG_RES
END

InspMenu  MENU
BEGIN
 POPUP "&Edit"
```

```
    BEGIN
     MENUITEM "Cu&t\tShift+Del",  EDIT_CUT
     MENUITEM "&Copy\tCtrl+Ins", EDIT_COPY
     MENUITEM "&Paste\tShift+Ins", EDIT_PASTE
     MENUITEM "C&lear", EDIT_CLEAR
    END

   MENUITEM  "&Doit!", INSP_DOIT
   POPUP "&Inspect"
    BEGIN
     MENUITEM "&Variable", INSP_IVAR
     MENUITEM "&Key", INSP_IKEY
     MENUITEM "&Selection", INSP_ISEL
    END
END

BrowMenu  MENU
BEGIN
 MENUITEM "&Accept!", BR_ACCEPT
 POPUP "&Edit"
  BEGIN
   MENUITEM "Cu&t\tShift+Del", EDIT_CUT
   MENUITEM "&Copy\tCtrl+Ins", EDIT_COPY
   MENUITEM "&Paste\tShift+Ins", EDIT_PASTE
   MENUITEM "C&lear", EDIT_CLEAR
   MENUITEM SEPARATOR
   MENUITEM "Select &All\tCtrl+A", EDIT_SELALL
   MENUITEM "&Reformat\tCtrl+R", BR_REFORM
   MENUITEM SEPARATOR
   MENUITEM "D&elete Class", BR_DELCL, grayed
   MENUITEM "&Delete Method", BR_DELME, grayed
  END

 MENUITEM "&Doit!", INSP_DOIT
 MENUITEM "&Inspect!", INSP_ISEL

 POPUP "&Options"
  BEGIN
   MENUITEM "A&bout the class", BR_CABOUT, Grayed
   MENUITEM "&Make descendant", BR_CDES, Grayed
   MENUITEM SEPARATOR
   MENUITEM "&Class methods", BR_CMETH
   MENUITEM "&Object methods", BR_OMETH
   MENUITEM SEPARATOR
```

```
    MENUITEM "&Alphabetical", BR_ALPH
    MENUITEM "&Hierarchical", BR_HIER
    MENUITEM SEPARATOR
    MENUITEM "&ZoomEdit\tCtrl+Z", BR_ZOOM
    MENUITEM "&Refresh Class List", BR_REFRCL
   END

 POPUP "&Utility"
 BEGIN
  MENUITEM "&Implementors", WORK_IMP
  MENUITEM "&Senders", WORK_SYMSEND
  MENUITEM "&Global References", WORK_GLOSEND
  MENUITEM "&References", WORK_SEND
 END

 POPUP "&Templates"
  BEGIN
    MENUITEM "&do",  TEMP_DO
    MENUITEM "&if/then", TEMP_IF
    MENUITEM "if/&else", TEMP_IFEL
    MENUITEM "&block", TEMP_BLOCK
    MENUITEM "&select/case", TEMP_CASE
    MENUITEM "&loop", TEMP_LOOP
    MENUITEM SEPARATOR
    MENUITEM "&New method", TEMP_NMETH
  END
END

WorkMenu  MENU
BEGIN
 POPUP "&File"
   BEGIN
    MENUITEM "&Run...",  WORK_RUN
    MENUITEM "&Edit...",  WORK_EDIT
    MENUITEM "&Load...",  WORK_LOAD
    MENUITEM "&Snapshot", WORK_SNAP
   END

 POPUP "&Edit"
   BEGIN
    MENUITEM "Cu&t\tShift+Del",  EDIT_CUT
    MENUITEM "&Copy\tCtrl+Ins", EDIT_COPY
    MENUITEM "&Paste\tShift+Ins", EDIT_PASTE
    MENUITEM "C&lear", EDIT_CLEAR
```

```
    MENUITEM SEPARATOR
    MENUITEM "Select &All\tCtrl+A", EDIT_SELALL
   END

  MENUITEM "&Doit!", INSP_DOIT

  MENUITEM "&Inspect!", INSP_ISEL

  MENUITEM "&Browse!", WORK_BROWSE

  MENUITEM "&Cleanup!", WORK_CLEAN

  MENUITEM "&Show Room!", WORK_ROOM

  POPUP "&Utility"
   BEGIN
    MENUITEM "&Implementors", WORK_IMP
    MENUITEM "&Senders", WORK_SYMSEND
    MENUITEM "&Global References", WORK_GLOSEND
    MENUITEM "&References", WORK_SEND
    MENUITEM SEPARATOR
    MENUITEM "&Clear Display", WORK_CLSDISP
   END

  POPUP "&Templates"
   BEGIN
    MENUITEM "&do",  TEMP_DO
    MENUITEM "&if/then", TEMP_IF
    MENUITEM "if/&else", TEMP_IFEL
    MENUITEM "&block", TEMP_BLOCK
    MENUITEM "&select/case", TEMP_CASE
    MENUITEM "&loop", TEMP_LOOP
   END
  MENUITEM "Demos!", WORK_DEMO
END

FileEditMenu  MENU
BEGIN
  POPUP "&File"
   BEGIN
    MENUITEM "&New", FILE_NEW
    MENUITEM "&Open...", FILE_OPEN
    MENUITEM "&Insert File...", FILE_READ
    MENUITEM "&Save", FILE_SAVE
```

```
    MENUITEM "Save &As...", FILE_SAVEAS
   END

  POPUP "&Edit"
   BEGIN
    MENUITEM "Cu&t\tShift+Del",  EDIT_CUT
    MENUITEM "&Copy\tCtrl+Ins", EDIT_COPY
    MENUITEM "&Paste\tShift+Ins", EDIT_PASTE
    MENUITEM "C&lear", EDIT_CLEAR
    MENUITEM SEPARATOR
    MENUITEM "Select &All\tCtrl+A", EDIT_SELALL
   END
  MENUITEM "&Doit!", INSP_DOIT
  MENUITEM "&Inspect!", INSP_ISEL
END
```

APPENDIX B.ACTOR CODE FOR DMWINDOW CLASS

The following listings are the DMWindow class code that was either created or modified for the implementation of this thesis.

```
/* GLAD Window for data manipulation interaction */!!

inherit(MyWindow, #DMWindow, #(dbSchema /*meta data of opened db*/
prevObj  /*previously selected
        object if any */
selObj   /*currently selected
        object if any*/
colorTable /*available colors for
          shading*/
rectSize /*width & height of object rectangle
         expressed in Point*/
objMoved /*true if object is
       dragged*/
prevCurPt /*previous cursor point
         while object is dragged*/
boundRect /*bounding box that surrounds all objects.
          used for scroll bars & setting windoworg*/
logOrg  /*origin of logical coordinate,
        mapped to device coord (0,0)*/
hScroll /*true if there is HScrollBar*/
vScroll /*true if there is VscrollBar*/
bDDEAcknowledge /* has DDE initiation
been acknowledged? */
mbdsDB /* Is the current DB an MBDS
database? */
hSockets /* Handle of Socket interface window */
haveData /* have the results of the retrieve come back? */
awaitingData /* Are we waiting for data from MBDS */
waitDialog /* Dialog box which shows while waiting for data */
dbName /* Name of the database */
qwin /* query window */), 2, nil)!!



now(DMWindowClass)!!
```

```
now(DMWindow)!!

/* comment */
Def  returnQuery(self | win,aStr, aCommand )
{
   aStr := formulateRetrieve(self);
   aCommand := addAtom(self, lP(aStr));
   Call PostMessage(hSockets, WM_DDE_REQUEST, hWnd, pack(CF_TEXT,
aCommand));

     waitDialog := new(Dialog);
     runModal(waitDialog, DATAWAIT, self);
    haveData := "YES";

   if haveData = "YES"
     win := new(ListMemWindow,self,"GladLMMenu","BROWSE:
"+name(selObj),nil);
     addWindow(selObj,win);
     start(win,selObj);
   endif;
  }!!

Def QueryMembers(self | queryDialog aStr aCommand win)
{
 if not(selObj)
  errorBox("ERROR!","No object is selected")
 else
  if mbdsDB

     awaitingData := true;
     queryDialog := new(InputDialog," Queries to
MBDS","Query:","FIRST@[");
     if runModal(queryDialog,INPUT_BOX,ThePort)==IDOK
       aStr:=getText(queryDialog)
     endif;

     aStr := "020"+  aStr;
     aCommand := addAtom(self, lP(aStr));
     Call PostMessage(hSockets, WM_DDE_REQUEST, hWnd, pack(CF_TEXT,
aCommand));


     waitDialog := new(Dialog);
```

```
    runModal(waitDialog, DATAWAIT, self);
   else haveData := "YES";
   endif;
   if haveData = "YES"
    win := new(ListMemWindow,self,"GladLMMenu","BROWSE:
"+name(selObj),nil);
    addWindow(selObj,win);
    start(win,selObj);
   endif;
  endif
}       !!

!!

/*RetCommon queries into the database*/
Def qRetCommon(self)
{
 errorBox("query Retrieve Common","")
} !!

/*Delete queries into the database*/
Def qDelete(self)
{
 errorBox("query Delete","")
} !!

/*Update queries into the database*/
Def qUpdate(self)
{
 errorBox("query Update","")
} !!

/*Insert queries into the database*/
Def qInsert(self)
{
 errorBox("query Insert","")
} !!

/* Formulate a retrieve request to get selected MBDS object's data */
Def  formulateRetrieve(self | aStr, attribs)
{
 aStr := new(String, 100);
```

```
      aStr := "020" + asUpperCase(dbName) + "@" + "[RETRIEVE(TEMP=" +
name(selObj) + ")(";
     attribs := new(String, 50);
     attribs := "";
     do(attributes(selObj),
        {using(attr)
          attribs := attribs + asUpperCase(attr[NAME]) + ",";
        }
       );
     aStr := aStr + attribs + "&";
     aStr := subString(aStr, 0, indexOf(aStr, '&', 0) - 1);
     aStr := aStr + ")BY ";
     aStr := aStr + subString(attribs, 0, indexOf(attribs, ',', 0)) + "]";
     ^asciiz(aStr);
    }!!

    /* Tell MBDS to load the appropriate database */
    Def  loadMBDSDatabase(self, dbName | aStr, aCommand)
    {
     aStr := delete(dbName, 0, 4);
     aStr := leftJustify(aStr);
     aStr := asUpperCase(rightJustify(aStr));
     aStr := "010" + aStr;
     aCommand := addAtom(self, lP(asciiz(aStr)));
       Call  PostMessage(hSockets,  WM_DDE_REQUEST,  hWnd,  pack(CF_TEXT,
aCommand));
    }!!

    /* Handle Dynamic Data Exchange acknowledgments */
    Def  WM_DDE_ACK(self, wp, lp | aServer, aTopic)
    {
     aServer := Call GlobalFindAtom(lP(asciiz("Sockets")));
     aTopic := Call GlobalFindAtom(lP(asciiz("Database")));
     deleteAtom(self, aServer);
     deleteAtom(self, aTopic);
     hSockets := wp;
     bDDEAcknowledge := true;
    }!!

    /* Get the latest data for a MBDS object */
    Def getMembers(self | aStr aCommand)
    {
     aStr := formulateRetrieve(self);
```

```
    aCommand := addAtom(self, lP(aStr));
      Call PostMessage(hSockets, WM_DDE_REQUEST, hWnd, pack(CF_TEXT,
aCommand));
   }!!

   /* Handle Socket interfaces's WM_DDE_DATA messages here */
   Def WM_DDE_DATA(self, wp, lp | mbdsCommand, aStr, objFile, delFile)
   {
    aStr := getAtom(self, high(lp));
    deleteAtom(self, high(lp));
    mbdsCommand := subString(aStr, 0, 3);
    select

     case mbdsCommand = "020"
      objFile := new(File);
      delFile := new(File);
      setName(delFile, memberFile(selObj));
      delete(delFile);
      setName(objFile, "qresults.fil");
      reName(objFile, memberFile(selObj));
      setHaveMembers(selObj, true);
      haveData := "YES";
      if awaitingData
       end(waitDialog, 0);
      endif;
     endCase;

     default
      haveData := "ERROR";
      if awaitingData
       end(waitDialog, 0);
      endif;
      errorBox("ERROR", delete(aStr, 0, 3));

    endSelect;
   }!!

   /* Return the string that the atom references */
   Def getAtom(self, atom | bufStr, aStr)
   {
    bufStr := new(String, 100);
    Call GlobalGetAtomName(atom, lP(bufStr), 100);
    aStr := removeNulls(getText(bufStr));
```

```
  freeHandle(bufStr);
  ^aStr
 }!!

 /* Delete or decrease the count on an atom */
 Def  deleteAtom(self, param)
 {
  if ((param >= 0xC000) and (param <= 0xFFFF))
   ^Call GlobalDeleteAtom(param);
  endif;
  ^nil
 }!!

 /* Create a new atom or increment the count of one that already exists */
 Def  addAtom(self, param)
 {
  ^Call GlobalAddAtom(param);
  freeHandle(param);
 }!!

 /* Initiate Dynamic Data Exchange with the Socket interface */
 Def  initDDE(self, reason | aServer, aTopic)
 {
  bDDEAcknowledge := false;
  aServer := addAtom(self, lP(asciiz("Sockets")));
  aTopic := addAtom(self, lP(asciiz("Database")));
     Call  SendMessage(0xFFFFL,  WM_DDE_INITIATE,  hWnd,  pack(aServer,
aTopic));
  if not(bDDEAcknowledge)
   if reason = "initDDE"
    errorBox("ERROR","Unable to initiate DDE with the Socket Interface");
   endif;
   ^false;
  else
   ^true;
  endif;
 }!!

 Def reDraw(self, obj | hdc)
 {
  hdc := getContext(self);
  setLPtoDP(self,hdc);
  display(self,obj,hdc);
```

```
 releaseContext(self,hdc)
}!!  !!

/*gets the filename from the name listed in the listbox*/
Def getGLADfilename(self ,selDbl tmpStr)
{
 tmpStr := new(String,30);
 tmpStr := "";
 do(selDb, {using(elem)
            if elem <> ' '
              tmpStr := tmpStr + asString(elem)
            endif  });
 ^subString(tmpStr,0,7) + ".sch"
}           !!     !!     !!

/*returns the currently selected object*/
Def selObj(self)
{
 ^selObj
}!!

/* draws an object on the window using
   the hdc display context */
Def display(self,obj,hdc | objName, objRect,
                      hBrush, hPen, hOldBrush, hOldPen)
{
 eraseRect(self,obj,hdc); /*first erase it*/
 /*select the color brush for filling
   used with Rectangle (via draw) */
 hBrush := Call CreateSolidBrush(color(obj));
 /*set bkcolor for shading with DrawText*/
 Call SetBkColor(hdc,color(obj));
 hOldBrush := Call SelectObject(hdc,hBrush);
 objRect := rect(obj);
 if obj.thickBorder /*draw it with a thick border*/
   hPen   := Call CreatePen(0,5,Call GetTextColor(hdc));
   hOldPen:= Call SelectObject(hdc,hPen);
   draw(objRect,hdc);
   Call SelectObject(hdc,hOldPen);/*restore the dc*/
   Call DeleteObject(hPen)
 else
   draw(objRect,hdc) /*with a reg. border*/
 endif;
 if nesting(obj) /*draw the inner box if it is a nested object*/
```

```
  draw(nestedRect(obj),hdc)
 endif;

 objName := name(obj);
 Call DrawText(hdc,lP(objName),-1,objRect,
          DT_CENTER bitOr DT_VCENTER
          bitOr DT_SINGLELINE);
 Call SelectObject(hdc,hOldBrush);
 Call DeleteObject(hBrush);
 freeHandle(objName)
}   !!

/*erase the region a little larger than object
 rectangle in case it is displayed with a thick
 border*/
Def eraseRect(self,obj,hdc | hBrush tmpRect)
{
 tmpRect := copy(rect(obj));
 hBrush := Call CreateSolidBrush(WHITE_COLOR);
 Call FillRect(hdc,inflate(tmpRect,5,5),hBrush);
 Call DeleteObject(hBrush)
}   !!

/*clientRect for DMWindow ignores the scroll bars
 if present*/
Def clientRect(self | cRect, incr)
{
 cRect := clientRect(self:WindowsObject);
 if hScroll
  incr := Call GetSystemMetrics(3);/*SM_CYHSCROLL*/
  setBottom(cRect, bottom(cRect)+incr-1)
 endif;
 if vScroll
  incr := Call GetSystemMetrics(2);/*SM_CXVSCROLL*/
  setRight(cRect, right(cRect)+incr-1)
 endif;
 ^cRect
}  !!

/*move vert scroll bar down for incr amount*/
Def moveDownVScroll(self, incr | newy)
{
 newy := min(y(logOrg) + incr,
```

```
        bottom(boundRect) - height(clientRect(self)));
 /*adjust newy so it won't go beyond boundRect*/
 setLogOrg(self,x(logOrg),newy);
 setScrollPos(self,SB_VERT,newy);
 repaint(self)
}  !!

/*move vert scroll bar up for incr amount*/
Def moveUpVScroll(self, incr | newy)
{
/*adjust newy so it won't go beyond boundRect*/
 newy := max(y(logOrg) - incr,top(boundRect));
 setLogOrg(self,x(logOrg),newy);
 setScrollPos(self,SB_VERT,newy);
 repaint(self)
}

  !!

/*check if any of four constraints is violated
 if so adust accordingly */
Def checkForViolation(self | cRect)
{
 cRect := clientRect(self);
 if (x(logOrg)+right(cRect)) > right(boundRect)
   logOrg.x := max(left(boundRect),
               right(boundRect)-right(cRect));
   setRight(boundRect, max(x(logOrg)+right(cRect),
                       right(boundRect)))
 endif;

 if (y(logOrg)+bottom(cRect)) > bottom(boundRect)
   logOrg.y := max(top(boundRect),
               bottom(boundRect)-bottom(cRect));
   setBottom(boundRect,max(y(logOrg)+bottom(cRect),
                       bottom(boundRect)))
 endif;

 if x(logOrg) < left(boundRect)
   setLeft(boundRect, min(left(boundRect),
                      right(boundRect)-right(cRect)));
   logOrg.x := left(boundRect)
 endif;
```

```
 if y(logOrg) < top(boundRect)
  setTop(boundRect, min(top(boundRect),
                bottom(boundRect)-bottom(cRect)));
  logOrg.y := top(boundRect)
 endif
}  !!

/*clientRect has changed. process only if the
 change is not by the changes in scroll
 bars, i.e. window size really changed */
Def reSize(self,wp,lp)
{
 adjBoundRect(self);/*always be adjusted*/
 setScrollRanges(self);
 repaint(self)

}     !!

/*move horz scroll bar right for incr amount*/
Def moveRightHScroll(self, incr | newx)
{
 newx := min(x(logOrg) + incr,
         right(boundRect)-width(clientRect(self)));
 /*adjust newx so it won't go beyond boundRect*/
 setLogOrg(self,newx,y(logOrg));
 setScrollPos(self,SB_HORZ,newx);
 repaint(self)
}
   !!

/*move horz scroll bar left for incr amount*/
Def moveLeftHScroll(self, incr | newx)
{
 newx := max(x(logOrg) - incr,left(boundRect));
 /*adjust newx so it won't go beyond boundRect*/
 setLogOrg(self,newx,y(logOrg));
 setScrollPos(self,SB_HORZ,newx);
 repaint(self)
}

   !!

Def hThumbPos(self, lp | newx)
```

```
{
 newx := asInt(lp);
 setLogOrg(self,newx,y(logOrg));
 setScrollPos(self,SB_HORZ,newx);
 repaint(self)
} !!

Def vThumbPos(self, lp | newy)
{
 newy := asInt(lp);
 setLogOrg(self,x(logOrg),newy);
 setScrollPos(self,SB_VERT,newy);
 repaint(self)
} !!

Def downPage(self,lp)
{
 moveDownVScroll(self,height(clientRect(self)))
} !!

Def upPage(self,lp)
{
 moveUpVScroll(self,height(clientRect(self)))
} !!

Def rightPage(self,lp)
{
 moveRightHScroll(self,width(clientRect(self)))
} !!

Def leftPage(self,lp)
{
 moveLeftHScroll(self,width(clientRect(self)))
} !!

Def rightArrow(self,lp)
{
  moveRightHScroll(self,asInt(0.25*x(rectSize)))
} !!

Def downArrow(self,lp)
{
 moveDownVScroll(self,asInt(0.5*y(rectSize)))
```

```
} !!

Def upArrow(self,lp)
{
 moveUpVScroll(self,asInt(0.5*y(rectSize)))
} !!

Def leftArrow(self,lp)
{
 moveLeftHScroll(self,asInt(0.25*x(rectSize)))
}  !!

/*convert device pt (DP) to logical pt (LP)*/
Def dPtoLP(self,aPt)
{
 ^point(aPt.x + logOrg.x, aPt.y + logOrg.y)
} !!

/*set the logical coord. origin*/
Def setLogOrg(self,x,y)
{
 logOrg.x := x;
 logOrg.y := y
} !!

/*logOrg is now mapped to device coord. (0,0)*/
/*need to pass hdc since there could be two disp context
allocated to this window at one time */
Def setLPtoDP(self, hdc)
{
 Call SetWindowOrg(hdc,x(logOrg),y(logOrg))
}  !!

Def setScrollRanges(self | xmin,ymin,xmax,ymax)
{
 xmin:=left(boundRect);
 ymin:=top(boundRect);
 cRect := clientRect(self);
 if (xmax:=right(boundRect)-width(cRect)) < xmin
   xmax := xmin
 endif;
 if (ymax:=bottom(boundRect)-height(cRect)) < ymin
   ymax := ymin
```

```
 endif;
 hScroll := xmin < xmax;
 vScroll := ymin < ymax;
 setScrollRange(self,SB_HORZ,xmin,xmax);
 setScrollRange(self,SB_VERT,ymin,ymax)
}  !!

Def adjBoundRect(self |tmpRect)
{
 tmpRect:=copy(boundRect);
 boundRect:=copy(rect(first(dbSchema)));
 do(dbSchema,
    {using(obj | objRect)
     objRect := rect(obj);
     setTop   (boundRect,min(top(objRect),top(boundRect)));
     setLeft  (boundRect,min(left(objRect),left(boundRect)));
     setBottom(boundRect,max(bottom(objRect),bottom(boundRect)));
     setRight (boundRect,max(right(objRect),right(boundRect)));
    } );
 /*changes boundRect accordingly per violation*/
 checkForViolation(self);
 if tmpRect = boundRect
  ^nil
 endif
}    !!

Def start(self, databaseName | dosFilename, DDEInitiated)
{
 dbName := databaseName;
 mbdsDB := false;
 if subString(dbName, 0, 4) = "MBDS"
  mbdsDB := true;
  if not(initDDE(self, "socketActive?"))
   exec("sockets.exe");
  endif;
  if (DDEInitiated := initDDE(self, "initDDE"))
   loadMBDSDatabase(self, dbName);
   dbName := delete(dbName, 0, 5);
  endif;
 endif;
 if not(mbdsDB) or DDEInitiated
  dosFilename := getGLADfilename(self,dbName);
  loadSchema(self, dosFilename);
  adjBoundRect(self);
```

```
   setLogOrg(self, /*center of bRect to center of cRect*/
     left(boundRect)+asInt(0.5*(width(boundRect)-width(clientRect(self)))),
      top(boundRect)+asInt(0.5*(height(boundRect)-height(clientRect(self)))));
   setScrollRanges(self);
   setScrollPos(self,SB_HORZ,x(logOrg));
   setScrollPos(self,SB_VERT,y(logOrg));
   show(self,1);
  endif;
}       !!

/*set the width and height of object rectangle*/
Def setObjRectSize(self | tm, wd, ht)
{
  tm := new(Struct,32);
  Call GetTextMetrics(hDC:=Call GetDC(hWnd),tm);
  wd:= 14*asInt(wordAt(tm,10));
  ht:=  4*asInt(wordAt(tm, 0));
  rectSize := point(wd,ht);
  Call ReleaseDC(hWnd,hDC)
}    !!

/*mouse is dragged while left button is pressed.
  move obj if mouse is in it */
Def mouseMoveWithLBDn(self,wp,point | aLPt)
{
  if selObj
    objMoved := true;
    eraseRect(self,selObj,hDC);
    aLPt:= dPtoLP(self,point); /*convert to DP to LP*/
    setNewRect(selObj,aLPt,prevCurPt);
    prevCurPt:= aLPt;
    display(self,selObj,hDC)
  endif
}       !!

Def initMenuID(self)
{
  menuID := %Dictionary( 1->#describe
                         2->#expand
                         3->#listMembers
                         4->#oneMember
                         5->#addMember
                         6->#deleteMember
```

```
                    7->#modifyMember
                    8->#query
                    9->#showConnection
                    950->#help
                    11->#close
                    21->#describe
                    22->#qInsert
                    23->#qRetrieve
                    24->#qUpdate
                    25->#qDelete
                    26->#qRetCommon )
}

      !!

Def showConnection(self | aConnWin)
{
 errorBox("Show Connection","")
}   !!

Def expand(self |win )
{
 if not(selObj)
  errorBox("ERROR!","No Object is selected")
 else
  if not(nesting(selObj))
   errorBox("ERROR!","Selected Object is not a nested object")
  else
   win := new(NestDMWindow,self,"GladDMlMenu",
            "SubClasses of: "+name(selObj),nil);
   addWindow(selObj,win);
   start(win,selObj,colorTable)
  endif
 endif
}   !!

/*open oneMemWin for the selected object*/
Def oneMember(self | oneMemWin, gotMembers)
{
 if not(selObj)
  errorBox("ERROR", "No Object Selected")
 else
  if mbdsDB and not(haveMembers(selObj))
```

```
    getMembers(self);
    awaitingData := true;
    waitDialog := new(Dialog);
    runModal(waitDialog, DATAWAIT, self);
   else haveData := "YES";
   endif;
   if haveData = "YES"
    oneMemWin := new(DisplayOneWindow,self,"GladOMMenu",
                "DISPLAY: "+name(selObj),nil);
    addWindow(selObj,oneMemWin);
    start(oneMemWin,selObj,0);
   endif;
 endif
}
      !!

/*count the number of the describe window opened*/
Def countOpnDscrbWin(self)
{
 ^size(extract(dbSchema,{using(obj) obj.aDscrbWin}))
} !!

/*initialize the color table.  this method
 is called from the new method*/
Def init(self)
{
 colorTable := new(ColorTable,10);
 set(colorTable);
 logOrg :=0@0;  /*need dummy assigment,so initial call
            to checkForViolation via reSize works*/
 boundRect:=new(Rect);
 setObjRectSize(self);
 init(self:MyWindow)
}      !!

Def rButtonRelease(self,wp,point | tmpObj)
{
 if (tmpObj := objSelected(self,dPtoLP(self,point)))
  /*an object is clicked with rbutton*/
   if tmpObj <> selObj
    if color(tmpObj) == WHITE_COLOR
      errorBox("Wrong Button??",
            "Use LEFT button to select an object")
```

```
        else
          errorBox("E R R O R",
                "RIGHT button clicked object is not"+CR_LF+
                "the selected (bold-lined) object")
        endif
       else /* = selObj */
        closeOpenWindows(selObj);
        if not(referenced(selObj))
         /*unshade it if not referenced by other objects*/
         avail(colorTable,color(selObj));
         setColor(selObj,WHITE_COLOR)
        endif;
        /*now unselect it*/
        regBorder(selObj);
        hDC := getContext(self);
        setLPtoDP(self,hDC);
        display(self,selObj,hDC);
        releaseContext(self,hDC);
        selObj := nil
       endif
      endif
    }      !!

    /*list the members (ie instances) of the
     selected object*/
    Def listMembers(self | win, gotMembers)
    {
     if not(selObj)
      errorBox("ERROR!","No object is selected")
     else
      if mbdsDB and not(haveMembers(selObj))
       getMembers(self);
       awaitingData := true;
       waitDialog := new(Dialog);
       runModal(waitDialog, DATAWAIT, self);
      else haveData := "YES";
      endif;
      if haveData = "YES"
                win    :=    new(ListMemWindow,self,"GladLMMenu","BROWSE:
"+name(selObj),nil);
       addWindow(selObj,win);
       start(win,selObj);
      endif;
```

```
 endif
}       !!

/*queries the database*/
Def query(selfl db so aStr qwin win aCommand)
{
 if not(selObj)
   errorBox("ERROR", "No Object Selected")
  else
   if mbdsDB
   awaitingData := true;
   so:= new(String, 50);
   db:= new(String, 50);
   so:= name(selObj);
   db:= asUpperCase(dbName);
   qwin := new(QueryWindow,self,"QueryGlad",
                "Glad to MBDS Queries OF: "+name(selObj),nil);

   start(qwin,db,so,hSockets,selObj,self);


 /*   aStr := formulateRetrieve(self);
   aCommand := addAtom(self, lP(aStr));
   Call PostMessage(hSockets, WM_DDE_REQUEST, hWnd, pack(CF_TEXT,
aCommand));

     waitDialog := new(Dialog);
     runModal(waitDialog, DATAWAIT, self);
   else haveData := "YES";*/
   endif;
 /*  if haveData = "YES"
     win := new(ListMemWindow,self,"GladLMMenu","BROWSE:
"+name(selObj),nil);
     addWindow(selObj,win);
     start(win,selObj);
   endif;*/
  endif
}            !!

Def help(selflaStr)
{aStr :=asciiz("Data Manipulation Window");
  pcall(Lib.procs[#GUIDANCESETCONTEXT],HGuide,
  lP(aStr),1);
```

```
 freeHandle(aStr);
} !!!!

/*describe the structure of the selected object*/
Def describe(self | describeWin)
{
 if not(selObj)
   errorBox("ERROR", "No Object Selected")
  else
   describeWin := new(DescribeWindow,self,nil,
                "STRUCTURE OF: "+name(selObj),nil);
   addWindow(selObj,describeWin);
   start(describeWin,selObj)
  endif;
}             !!

/*left button is released*/
Def lButtonRelease(self,wp,point| aLPt)
{
 select
   case selObj and not(objMoved)
   /*an object was not moved, so select it*/
   is
     if prevObj /*unbold the bolded border*/
       regBorder(prevObj);
       /*unshade it if has no opened windows
         and not referenced by other objects*/
       if not( anyOpenWindow(prevObj) or referenced(prevObj) )
         avail(colorTable,color(prevObj));
         setColor(prevObj,WHITE_COLOR)
       endif;
       display(self,prevObj,hDC)
     endif;
     if color(selObj) = WHITE_COLOR
       /*not referenced in another's describe window,
         so assign it a color*/
       setColor(selObj,nextBrushColor(colorTable))
     endif;
     thickBorder(selObj);
     display(self,selObj,hDC)
   endCase

   case selObj and objMoved
    /*an object was just moved, so don't select it*/
```

```
    /*adjust boundRect and scroll bars accordingly*/
    is
     display(self,selObj,hDC);
     selObj := prevObj;
     if adjBoundRect(self)
       setScrollRanges(self);
       setScrollPos(self,SB_HORZ,x(logOrg));/*need these when*/
       setScrollPos(self,SB_VERT,y(logOrg));/*bars reappear*/
       repaint(self)
     endif
   endCase
 endSelect;
 releaseContext(self,hDC)
}                    !!

/*left button is pressed; check if the cursor is within
 the object rectangle. If yes get ready to move or
 select it*/
Def lButtonDown(self,wp,point | aLPt)
{
 objMoved := nil;
 if selObj
   /*remember it if some object is currently selected*/
   prevObj := selObj
 endif;
 aLPt:= dPtoLP(self,point);
 if (selObj := objSelected(self,aLPt))
   prevCurPt := aLPt
 endif;
 hDC := getContext(self);
 setLPtoDP(self,hDC)
}                    !!

/*detects whether the cursor is in the object rect*/
Def objSelected(self,cursorPt)
{
 do (dbSchema,{using(obj)
            if containedIn(obj,cursorPt)
              ^obj /*return the selected obj*/
            endif });
 ^nil
}    !!

/*draws the diagram. called by the show method
```

```
 via update method which sends WM_PAINT */
Def paint(self,hdc)
{
 setLPtoDP(self,hdc);
 do (dbSchema, {using(obj) display(self,obj,hdc)})
}    !!

/*gets the meta data of db to be opened,
 initialize other instance variables*/
Def loadSchema(self, aSchemaFile | aFile, anObj)
{
 aFile := new(TextFile);
 setName(aFile,aSchemaFile);
 open(aFile,0); /*read-only*/
 dbSchema := new(OrderedCollection,10);
 anObj := new(GladObj);
 loop
 while get(anObj,aFile,rectSize)
  add(dbSchema,anObj);
  anObj := new(GladObj)
 endLoop;
 close(aFile)
}         !!

Def addMember(self)
{
 errorBox("add member","")
} !!

Def deleteMember(self)
{
 errorBox("delete member","")
} !!

Def modifyMember(self)
{
 errorBox("modify member","")
} !!
```

APPENDIX C. ACTOR CODE FOR QUERYWINDOW CLASS

The following listings are the QueryWindow class code that was either created or modified for the implementation of this thesis.

```
/* GLAD Window for ABDL queries to backend MBDS */!!


inherit(EditWindow, #QueryWindow, #(menuID /* menu idenities */
dbName /* database name */
selObjName /* selected object */
mbdsDB /* Is current DB an MDBS database */
boundRect/* */
logOrg /* */
hSockets /* handle of Socket interface window */
haveData /* have the results of the query come back */
awaitingData /* Are we waiting for data from MBDS */
waitDialog /*Dialog box which shows while waiting for data */
mbdsDB /* is the current DB an MBDS database */
bDDEAcknowledge /*has DDE initiation been acknowledged? */
selObj /* the selected object */
parentWin /* parent window DMWindow */
retquery /* is the query a retrieve */
ResultWin /* textwindow for retrieve results */
members/* */
TW/* is the template window */), 2, nil)!!

now(QueryWindowClass)!!

now(QueryWindow)!!

/* comment */
Def  Help(self)
{
errorBox("Help",
   "This program demonstrates simple ABDL" + CR_LF +
   "queries to a MBDS back-end" + CR_LF +
   "For a  example of each type of query," + CR_LF +
   "look under the template's menu item" + CR_LF +
   "and select the specific query.");
```

```
}!!

/*Delete queries into the database*/
Def tDelete(self)
{

 TW := new(TextWindow,ThePort,nil,
  "Format for an DELETE Query",&(20,20,400,100));
  show(TW,1);
  drawString(TW,"dbName@[DELETE((TEMP=selObj) and");
  eol(TW);
  drawString(TW, " (Attr1 = jones)) ]");

  setFocus(self);


} !!

/*Update queries into the database*/
Def tUpdate(self)
{

  TW := new(TextWindow,ThePort,nil,
  "Format for an Update Query",&(20,20,400,100));
  show(TW,1);
  drawString(TW,"dbName@[UPDATE((TEMP=selObj) and");
  eol(TW);
  drawString(TW," (Attr1 = jones)) <RANK = ENS>]");

  setFocus(self);

} !!

/*Retrieve queries into the database*/
Def tRetrieve(self)
{

   TW:= new(TextWindow,ThePort,nil,
  "Format for an RETRIEVE Query",&(20,20,400,150));
  show(TW,1);
  printString(TW,"dbName@[RETRIEVE((TEMP=selObjA)");
  eol(TW);
  printString(TW," and (Attr1 = jones))");
```

```
 eol(TW);
 printString(TW," (Attr1,Attr2,Attr3..)]");
 setFocus(self);


}!!

/*RetCommon queries into the database*/
Def tRetCommon(self)
{

 TW := new(TextWindow,ThePort,nil,
 "Format for an RETRIEVE COMMON Query",&(20,20,400,150));
 show(TW,1);
 printString(TW,"dbName@[RETRIEVE(TEMP=selObjA)(AttrA1,AttrA2,..)");
 eol(TW);
 printString(TW," COMMON (AttrA1,AttrB1)");
 eol(TW);
 printString(TW," RETRIEVE(TEMP= selObjB)(AttrB1,AttrB2,..)]");
 setFocus(self);




} !!

/*Insert queries into the database*/
Def tInsert(self)
{

 TW := new(TextWindow,ThePort,nil,
 "Format for an INSERT Query",&(20,20,400,100));
 show(TW,1);
 drawString(TW,"dbName@[INSERT(<TEMP,selObj>,");
 eol(TW);
 drawString(TW," <Attr1,jones>,<Attr2,LT>)]");

 setFocus(self);


} !!
```

```
/* Read the file named qwresult.dat and And show query results for retrieves*/

Def  RetResult(selfl f, line)
{
 break(self);


 f := new(TextFile);
 f.delimiter := CR_LF;  /* use our delimiter */
 setName(f, "qwresult.dat");
 open(f, 0);           /* read only */
 checkError(f);        /* any errors? */
 initWorkText(self);   /* clear the old text */
 showWaitCurs();       /* looping takes a while */
loop
 while line := readLine(f)
 printString(ResultWin,line);
 eol(ResultWin);
/*   add(workText, line);*/
 endLoop;


 showOldCurs();        /* all done */
 close(f);
 checkError(f);        /* just in case */
 invalidate(self);     /* redraw the screen */
}
!!

/* comment */
Def  formulateQuery(selflaStr)
{
aStr :=new(String,150);
selectAll(self);
xCopy(sel );
aStr:="";
aStr:=getClipText(self);


deleteSelText(self);
^asciiz(aStr);

 }!!
```

```
/* get the attributes from DMwindow for selobj */
Def  describe(self)
{
describe(parentWin);
}!!

/*list the members (ie instances) of the
  selected object*/
Def listMembers(self | win, gotMembers)
{


     queryMembers(self);
     awaitingData := true;
     waitDialog := new(Dialog);
     runModal(waitDialog, DATAWAIT, self);
     haveData := "YES";

    if haveData = "YES"

       if not(retquery)
     returnQuery(parentWin);
     else
     ResultWin:= new(TextWindow,ThePort,nil,
     "RESULTS of a Retrieve Query",nil);
     show(ResultWin,1);


     RetResult(self);
     endif;

    endif;

}          !!

/* Delete or decrease the count on an atom */
Def  deleteAtom(self, param)
{
 if ((param >= 0xC000) and (param <= 0xFFFF))
   ^Call GlobalDeleteAtom(param);
 endif;
 ^nil
}!!
```

```
/* Return the string that the atom references */
Def  getAtom(self, atom | bufStr, aStr)
{
 bufStr := new(String, 200);
 Call GlobalGetAtomName(atom, lP(bufStr), 200);
 aStr := removeNulls(getText(bufStr));
 freeHandle(bufStr);
 ^aStr
}!!

/* Handle Socket interfaces's WM_DDE_DATA messages here */
Def  WM_DDE_DATA(self, wp, lp | mbdsCommand, aStr, objFile, delFile)
{
 aStr := getAtom(self, high(lp));
 deleteAtom(self, high(lp));
 mbdsCommand := subString(aStr, 0, 3);
 select

  case mbdsCommand = "020"
   objFile := new(File);
    delFile := new(File);
   setName(delFile, "qwresult.dat");
   delete(delFile);

   setName(objFile, "qresults.fil");
   reName(objFile, "qwresult.dat");
   setHaveMembers(selObjName,false);
   haveData := "YES";
   if awaitingData
     end(waitDialog, 0);
   endif;
  endCase;

  default
   haveData := "ERROR";
   if awaitingData
     end(waitDialog, 0);
   endif;
   errorBox("ERROR", delete(aStr, 0, 3));

 endSelect;
}!!
```

```
/* Create a new atom or increment the count of one that already exists */
Def  addAtom(self, param ltempAtom)
{

 tempAtom:=Call GlobalAddAtom(param);


  ^tempAtom;
}!!

/* Get the latest data for a MBDS object */

Def queryMembers(self l aStr aCommand)
{

  aStr :=formulateQuery(self);

  aCommand := addAtom(self, lP(aStr));

     Call  PostMessage(hSockets,  WM_DDE_REQUEST,  hWnd,  pack(CF_TEXT,
aCommand));
}!!

Def start(self, qdbName, qselObj, handSocket,selObject,PWindow)
{
 initMenuID(self);
 show(self,1);
  hSockets := handSocket;
  selObjName:=selObject;
  dbName := qdbName;
  selObj := qselObj;
  parentWin :=PWindow;
  retquery:=false;
}       !!

/*RetCommon queries into the database*/
Def qRetCommon(self aStr,Rtc, attribs )
{
  retquery:=true;
  aStr:= new(String, 150);
  aStr:="020"+dbName+"@[RETRIEVE(TEMP="+selObj+")()";

  aStr:=aStr+"COMMON(,)RETRIEVE(TEMP= )()]";
```

```
 drawString(self,aStr);



} !!

/*Retrieve queries into the database*/
Def qRetrieve(selfl aStr,Ret, attribs)
{
 retquery:=true;
 aStr:= new(String, 150);
 aStr:="020"+dbName+"@[RETRIEVE(TEMP="+selObj+" )(";
 aStr := aStr + "    )] ";
 drawString(self,aStr);


}!!

/*Update queries into the database*/
Def qUpdate(selfl aStr, Upd, attribs)
{
 retquery:=false;
 aStr:= new(String, 150);
 aStr:="020"+dbName+"@[UPDATE((TEMP= "+selObj+" )";
 aStr := aStr + " and ( = ))<   =   >] ";
 drawString(self,aStr);
 }!!

Def command(self,wp,lp)
{/*only interprets the menu choice now*/
 if menuID[wp] and high(lp) = 1
   perform(self,menuID[wp]);
 else
   if menuID[wp]
     perform(self,menuID[wp])
   else command(self:EditWindow, wp, lp)
   endif;
 endif;
}!!

/*Insert queries into the database*/
```

```
Def qInsert(selfl aStr )
{
 retquery:=false;
 aStr:= new(String, 150);
 aStr:="020"+dbName+"@[INSERT(<TEMP,"+selObj+">,<";
 aStr := aStr + "ENAME,Nardi>,<EPHONE,x5555>)] ";
 drawString(self,aStr);

} !!

/*Delete queries into the database*/
Def qDelete(selfl aStr)
{
 retquery:=false;
 aStr:= new(String, 150);
 aStr:="020"+dbName+"@[DELETE((TEMP= "+selObj+" )";
 aStr := aStr + " and ( ))] ";
 drawString(self,aStr);


} !!

Def initMenuID(self)
{
 menuID := %Dictionary(
                    10->#Help
                    11->#close
                    12->#describe
                    21->#listMembers
                    22->#qInsert
                    23->#qRetrieve
                    24->#qUpdate
                    25->#qDelete
                    26->#qRetCommon
                    32->#tInsert
                    33->#tRetrieve
                    34->#tUpdate
                    35->#tDelete
                    36->#tRetCommon )
 } !!
```

LIST OF REFERENCES

1. Ruff, D., LCdr, USN, from:"The Advent of the Paperless Ship," *Naval Engineers Journal,* July 1988.

2. Duff, C., and others, *Actor Language Manual*, The Whitewater Group, Inc., 1989.

3. Newburger, B. , and others, "Introduction to Object-Oriented Programming", The Whitewater Group, Inc., 1989.

4. Duff, C., and others, *Actor Training Course Manual*, The Whitewater Group, Inc., 1988.

5. Jamsa, K., *Windows Programming Secrets,* Osborne McGraw-Hill, 1987.

6. Microsoft Corporation, *Microsoft Windows User's Guide,* Microsoft Press, 1987.

7. Wu, C.T. , Object-Oriented Programming through Actor, The Whitewater Group, Inc., in progress.

8. Zawis, J.A. , *Accessing Hierarchical databases via SQL Transactions in a Multi-Model Database System*, Masters Thesis, Naval Postgraduate School, Monterey, California, December 1987.

9. Emdi, B. , *The Implementation of a Network Codasyl-DML Interface for the Multi-Lingual database System*, Masters Thesis, Naval Postgraduate School, Monterey, California, December 1985.

10. Wong, A. , *Toward highly Portable Database System* , Masters Thesis, Naval Postgraduate School, Monterey, California, June 1986.

11. Hogan, T. R. ,*Interconnection of the Graphics Language for Database System to the Multi-Lingual, Multi-Model, Multi-Backend Database System Over an Ethernet Network,* Masters Thesis, Naval Postgraduate School, Monterey, California, December 1989.

12. Naval Postgraduate School Technical Report NPS52-88-050, *Implementation of Visual Database Interface Using an Object-Oriented Language*, by C. T. Wu and D. K. Hsiao, June 1988.

13. Fore, H. R., *Prototyping Visual Interface for Maintenance and Supply Databases*, Masters Thesis, Naval Postgraduate School, Monterey, California, June 1989.

14. Williamson, M. L., *An Implementation of a Data Definition Facility for the Graphics Language for Database*, Masters Thesis, Naval Postgraduate School, Monterey, California, December 1988.

INITIAL DISTRIBUTION LIST

	Number Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3. Office of Reasearch Adiministration Code 012 Naval Postgraduate School Monterey, California 93943-5002	1
4. Chairman, Computer Science Dept. Computer Science Department Naval Postgraduate School Monterey, California 93943-5002	1
5. Chief of Naval Research 800 N. Quincy Street Arlington, Virginia 22217-5000	1
6. Center for Naval Analyses 4401 Ford Avenue Arlington, Virginia 22302-0268	1
7. Naval Ocean Systems Center 271 Catalina Boulevard San Diego, California 92152	1
8. Curriculum Officer Computer Technology Program, code 37 Monterey, California 93943-5000	1

9. Professor C. Thomas Wu (Code 52Wq) — 25
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5000

10. Maria M. Jamini-Ramirez — 2
    Division Head
    MDS Division
    Data Systems Department
    Naval Weapons Station
    Concord, California 94520-5000

11. Robert Calogero — 2
    Director SEA CEL-PA
    Logistics Policy and Appraisal Division
    Naval Sea Systems Command
    Washington, D. C. 20362-5101

12. Clifford G. Geiger — 1
    Deputy Chief Engineer - Logistics
    Naval Sea Systems Command
    Washington, D. C. 20362-5101

13. LT. William G. A. Sympson — 5
    8B Sellers Road
    Annapolis, Maryland 21402

Thesis
S965 Sympson
c.1 Graphic interface for Attribute-Based Data Language queries from a personal computer to the Multi-Lingual, Multi-Model, Multi-Backend Database System over an ethernet network.